# A TRIDENT SCHOLAR PROJECT REPORT

NO. 264

## "MODEL BASED CONTROL OF COMBUSTION"


UNITED STATES NAVAL ACADEMY
ANNAPOLIS, MARYLAND

# REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188



| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATE COVERED |
|---|---|---|
| | 7 May 1999 | |

**4. TITLE AND SUBTITLE**
Model based control of combustion

**5. FUNDING NUMBERS**

**6. AUTHOR(S)**
Osburn, Nicholas G.

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
U.S. Naval Academy
Annapolis, MD

**8. PERFORMING ORGANIZATION REPORT NUMBER**
USNA Trident Scholar project report no. 264 (1999)

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

**11. SUPPLEMENTARY NOTES**
Accepted by the U.S. Trident Scholar Committee

**12a. DISTRIBUTION/AVAILABILITY STATEMENT**
This document has been approved for public release; its distribution is UNLIMITED.

**12b. DISTRIBUTION CODE**

13. ABSTRACT: This project investigates the use of mathematical modeling to design a closed-loop system for the control of combustion of a methane burner. Regulation of the air-to-fuel ratio for lean or rich combustion, depending on fuel efficiency or power requirements, is pivotal in applications such as internal combustion engines or fossil fuel power plants. Existing model-free approaches use means of limited capacity and require individual tuning through a cumbersome procedure of trial and error. The proposed design avoids the errors inherent in its model-free counterparts and, hence, is more capable of maintaining combustion at the desired air-to-fuel ratio. The model-based system operates by continuously measuring the quantity of carbon dioxide, carbon monoxide, or oxygen present within the exhaust of the combustion process. Using this measurement of the gases produced, a personal computer (PC) regulates the flow of fuel or air to the burner. The PC implements the required control algorithms, derived on the basis of a mathematical model of the combustion process, to control the burner accurately. The mathematical model is the product of system identification based exclusively on input-output measurement. The technology developed within this project will allow industries dependent upon combustion processes to control the progression and efficiency of such reactions more accurately. The final design provides a system capable of limiting the fuel feed or air flow into the combustion chamber to the narrow tolerances required for lean or rich combustion, despite fluctuations in load and varying environmental conditions.

**14. SUBJECT TERMS**
Model-Based Control, Air-to-Fuel Ratio, Combustion Control

**15. NUMBER OF PAGES**

**16. PRICE CODE**

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| | | | |

U.S.N.A.--Trident Scholar project report; no. 264 (1999)

# "MODEL BASED CONTROL OF COMBUSTION"

by

Midshipman Nicholas G. Osburn
United States Naval Academy
Annapolis, MD 21402

N. G. Osburn

Certification of Adviser Approval

Assistant Professor Kiriakos Kiriakidis
Department of Weapons and Systems Engineering

K. Kiriakidis

May 5, 1999

Professor Charles Rowell
Chemistry Department

Charles F Rowell

May 5, 1999

Acceptance for the Trident Scholar Committee

Professor Joyce E. Shade
Chair, Trident Scholar Committee

J. E. Shade

7 May 1999

USNA-1531-2

# ABSTRACT

This project investigates the use of mathematical modeling to design a closed-loop system for the control of combustion of a methane burner. Regulation of the air-to-fuel ratio for lean or rich combustion, depending on fuel efficiency or power requirements, is pivotal in applications such as internal combustion engines or fossil fuel power plants.

Existing model-free approaches use means of limited capacity and require individual tuning through a cumbersome procedure of trial and error. The proposed design avoids the errors inherent in its model-free counterparts and, hence, is more capable of maintaining combustion at the desired air-to-fuel ratio.

The model-based system operates by continuously measuring the quantity of carbon dioxide, carbon monoxide, or oxygen present within the exhaust of the combustion process. Using this measurement of the gases produced, a personal computer (PC) regulates the flow of fuel or air to the burner. The PC implements the required control algorithms, derived on the basis of a mathematical model of the combustion process, to control the burner accurately. The mathematical model is the product of system identification based exclusively on input-output measurement.

The technology developed within this project will allow industries dependent upon combustion processes to control the progression and efficiency of such reactions more accurately. The final design provides a system capable of limiting the fuel feed or air flow into the combustion chamber to the narrow tolerances required for lean or rich combustion, despite fluctuations in load and varying environmental conditions.

# KEYWORDS

Model-Based Control, Air-to-Fuel Ratio, Combustion Control

# NOMENCLATURE

**Analog:** A system with a continuous output given over a continuous time.

**C++:** A computer programming language.

**Closed Loop:** A system that incorporates feedback.

**Continuous Time:** A term used to describe a system that changes continually. The sensor and signal conditioning aspects of the project are such systems. Laplace transforms are used to evaluate these devices.

**Controller:** A device that is used to regulate a system's output to a desired response.

**Control Effort:** The output produced by a controller that is used to regulate the plant.

**Difference Equation:** An equation that describes the state of a system at any given time.

**Digital:** A term used to describe systems that operate over discrete time intervals at discrete values.

**Discrete Time:** A system that changes only at specific time intervals. The PC used within the project is such a system.

**Equivalence Ratio:** The chemically correct air/fuel ratio divided by the actual air/fuel ratio. Exact stoichiometric combustion possesses an equivalence ratio of one.

**Laplace Transform:** A transform that converts linear differential equations to algebraic polynomials.

**Linear:** A system that generates an output that is directly proportional to given parameters.

**Non-linear:** A system that generates an output that is not directly proportional to its parameters. Exponential, square, and inverse relationships are often found within this field.

**Open Loop:** A system that does not use feedback.

**Operating Point:** The equivalence ratio to which the system will be controlled.

**Operational Amplifier:** An electrical component used most often to amplify and invert voltage signals.

**Overshoot:** The percent by which the system exceeds the desired response.

**Plant:** The core of the system that the controller seeks to regulate.

**Pole:** A root of the denominator of a system's transfer function. The poles of a system characterize its behavior.

**Sampling Time:** The time interval of a discrete time, or digital, system.

**Stoichiometric Combustion:** Ideal combustion where all reactants are completely converted into the correct products.

**System Identification:** The process of producing a model that describes a system's operation.

**System Order:** The number of poles a system possesses.

**Time Constant:** The time the system requires to rise to 63% of its steady state response from a step input.

**Transfer Function:** An equation, using Z transforms or Laplace, transforms that describes a system's behavior by expressing it as a ratio of the system's input to its output.

**VisSim:** A computer simulation program that utilizes transfer functions to generate system responses.

**Z Transform:** A transform similar to the Laplace transform that is used to evaluate discrete time systems.

**Zero:** A root of the numerator of a system's transfer function.

**Z-Plane:** A coordinate system used to plot the poles and zeros of discrete time transfer functions. The location of these poles and zeros characterizes the system's behavior. For example, a system is stable if it does not possess any poles outside of the unit circle on the z-plane.

# TABLE OF CONTENTS

# I INTRODUCTION

Combustion processes are used within a wide variety of applications from powering the Navy's ships, automobiles, and aircraft to gas furnaces. However, with increasingly stringent requirements governing fuel efficiency and exhaust quality, contemporary control systems are having difficulty meeting these new specifications. They are limited in their abilities to regulate combustion processes to maintain the narrow margins of minimizing fuel consumption in any combustion process or maximizing power output in spark ignition engines.

Recently, in the United States, the trend toward industrial systems of greater efficiency has enjoyed considerable emphasis through methods designed to maximize productivity while minimizing the cost of wasteful processes. Applications that require combustion are of particular importance, being both intensive in their cost and the quality of exhaust they produce. Streamlining these processes has had the dual effect of increasing both the efficiency of plant operation, and improving the quality of flue gases expelled into the atmosphere. Within the last two decades, technology involving closed loop combustion control has tremendously aided these efforts[7]. Continued refinement of such feedback systems may be vitally important to both industry, which profits from the increased efficiency, and the population at large, which is dependent upon the quality of the environment.

Control systems are relevant to any application that requires combustion. From the operation of power plants to home furnaces and jet turbines, the exact regulation of the fuel-to-air ratio of the burner is of critical importance. Research has shown that for each percent reduction in excess oxygen in the reaction, combustion efficiency will rise by 1% [7]. Although existing technologies have been effective in improving the fuel-to-air ratio to near desirable levels, they are hampered by means of limited capacity when applications require closer tolerances. Roughly 10% of the process industries today can not be controlled to satisfactory levels using current technology [7]. Through the development of mathematical models of the combustion process, the controller created within this project is more capable of regulating the fuel-to-air ratio to stoichiometric levels despite variations in fuel and air quality within the burner or fluctuations in the load.

# PROBLEM STATEMENT

In ideal stoichiometric combustion, a fixed ratio of reactants governs the progress and output of the reaction. However, in real combustion, the reactant ratio is variable, and the efficiency of the reaction is dependent upon the quantity of each reactant present. The stoichiometric combustion of methane:

$$CH_4 + 2O_2 \rightarrow CO_2 + 2H_2O \tag{1.1}$$

requires one mole of methane for every two moles of oxygen present. As the ratio of methane to oxygen strays from the ideal, the combustion fuel efficiency varies. Since ideal combustion is impossible, the closer the reaction can be held slightly to the left of the stoichiometric ideal illustrated in Figure 1.1, the more efficient the process will be. "SFC" is the specific fuel consumption, and is a measure of efficiency, while "MEP" is the mean effective pressure, and is a measure of the power output in spark ignition engines. The top curve depicts the power output, and the bottom curve depicts combustion efficiency, which is defined by minimum specific fuel consumption. The stoichiometric ideal is represented by the dotted line. Therefore, mixtures of methane leaner than the stoichiometric ratio will fall to the left of the line, and richer mixtures to the right. The base of the bottom curve, which falls slightly to the left of the dotted line, represents the point of most efficient combustion [13]. The peak of the top curve represents maximum power output [13].

The project goal will be to construct a closed-loop control system that regulates the power and efficiency of methane combustion. The system concept is depicted in Figure 1.2. As shown in Figure 1.2, the initial control design mirrors systems currently used in industry, and uses a combination of digital and analog components driven by a Proportional-Integral (PI) controller. This controller functions by continuously measuring some of the exhaust gases of the burner, and by automatically adjusting the methane flow to control the reaction. The information gained in this stage of the project will be used to measure the performance of the proposed control design. The ultimate aim of the project will be realized in the creation of a controller, based upon a mathematical combustion model, capable of maintaining the burner at desired fuel-to-air ratios. A personal computer containing this controller will be inserted in place of the PI component in this final stage, and

will permit easy and rapid modification of the system when necessary. This combination of technologies, mathematical modeling and systems control design should provide more efficient fuel consumption and exhaust control than systems currently in use.

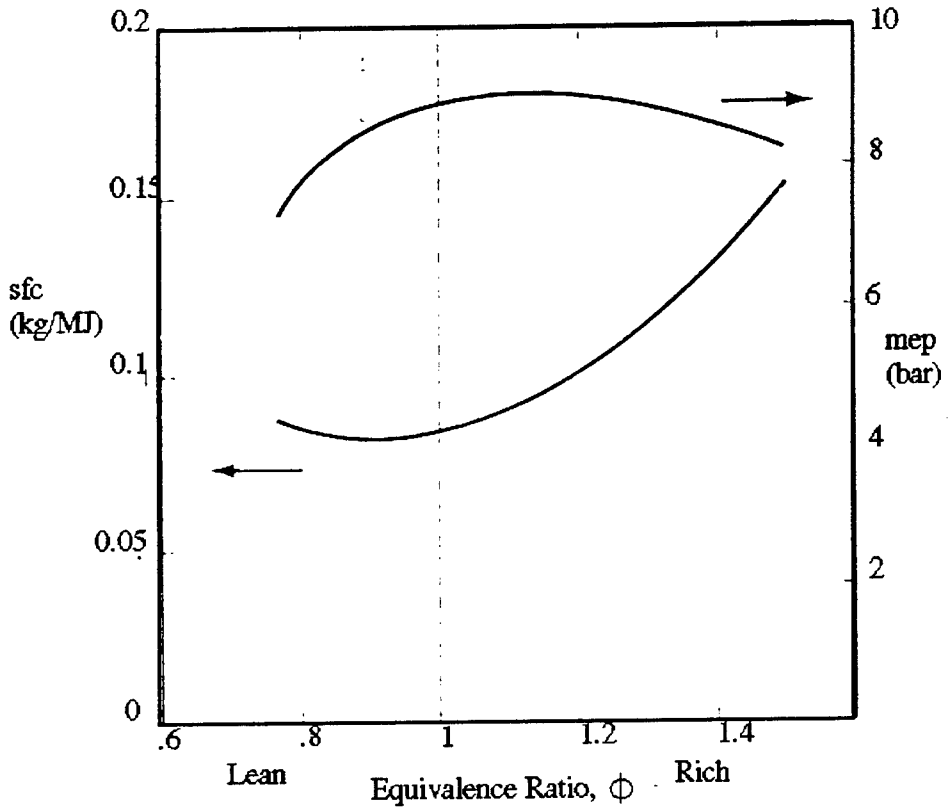



Figure 1.1 Generalized efficiency and power curves for combustion as described in the text [14]

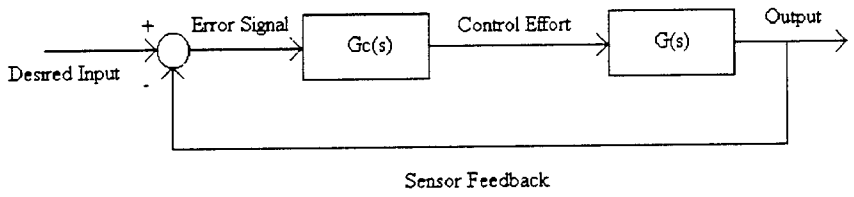



Figure 1.2 System Concept where Gc(s) is the controller (PC) and G(s) is the experimental apparatus

# CHEMISTRY

The reactants involved usually fall within two groups, fuels and oxidizers, and a reaction occurs when these compounds come in contact at high enough temperatures. The process of combustion is extremely complex, and is dependent upon many factors. Temperature, pressure, the concentration of reactants, and the concentration of products all affect the progression of the reaction. In equation (1.1), the simplified chemical equation for the combustion of methane is given. In reality, this process involves more than 10 different, yet interdependent, reactions.

Fuel Consumption Reactions

$$CH_4 + OH \Leftrightarrow CH_3 + H_2O$$
$$CH_4 + H \Leftrightarrow CH_3 + H_2$$
$$CH_4 + O \Leftrightarrow CH_3 + OH$$

Removal of the Methyl Radical

$$CH_3 + O \Leftrightarrow CH_2O + H$$

Thermal Decomposition and Reaction of Formaldehyde

$$CH_2O + M \Leftrightarrow CO + H_2 + M$$
$$CH_2O + OH \Leftrightarrow CHO + H_2O$$
$$CH_2O + O \Leftrightarrow CHO + OH$$

Reaction of CHO Radicals

$$CHO + O_2 \Leftrightarrow CO + HO_2$$

Reaction of Hydroperoxyl Radicals

$$HO_2 + OH(O) \Leftrightarrow O_2 + H_2O(OH)$$
$$HO_2 + H \Leftrightarrow 2OH$$

(1.2) [5]

Each of these reactions progresses at a different rate, and will reach transient equilibrium at varying times and under varying conditions. In addition, each of the reaction rates is exponentially dependent upon the temperature and activation energy of each step[5]. Although none of the reactions in equation (1.2) depict the production of carbon dioxide, they do portray the underlying complexity of

combustion. Because the combustion of methane involves so many separate reactions, and is dependent upon so many differing variables, simplifying the process is absolutely necessary before an effective control scheme can be derived. The experimental system is limited in its ability to detect gases, and to process information in a manner fast enough for control purposes. Therefore, gaseous measurements by the controller are restricted to CO or $CO_2$ in the burner's exhaust, and only the fuel entering the system is regulated. Although the fundamental progression of combustion is best characterized by exponential functions, a linear control scheme is utilized for simplicity. However, the final control algorithm is composed of three separate linear controllers that function to approximate the non-linear behavior of the combustion process.

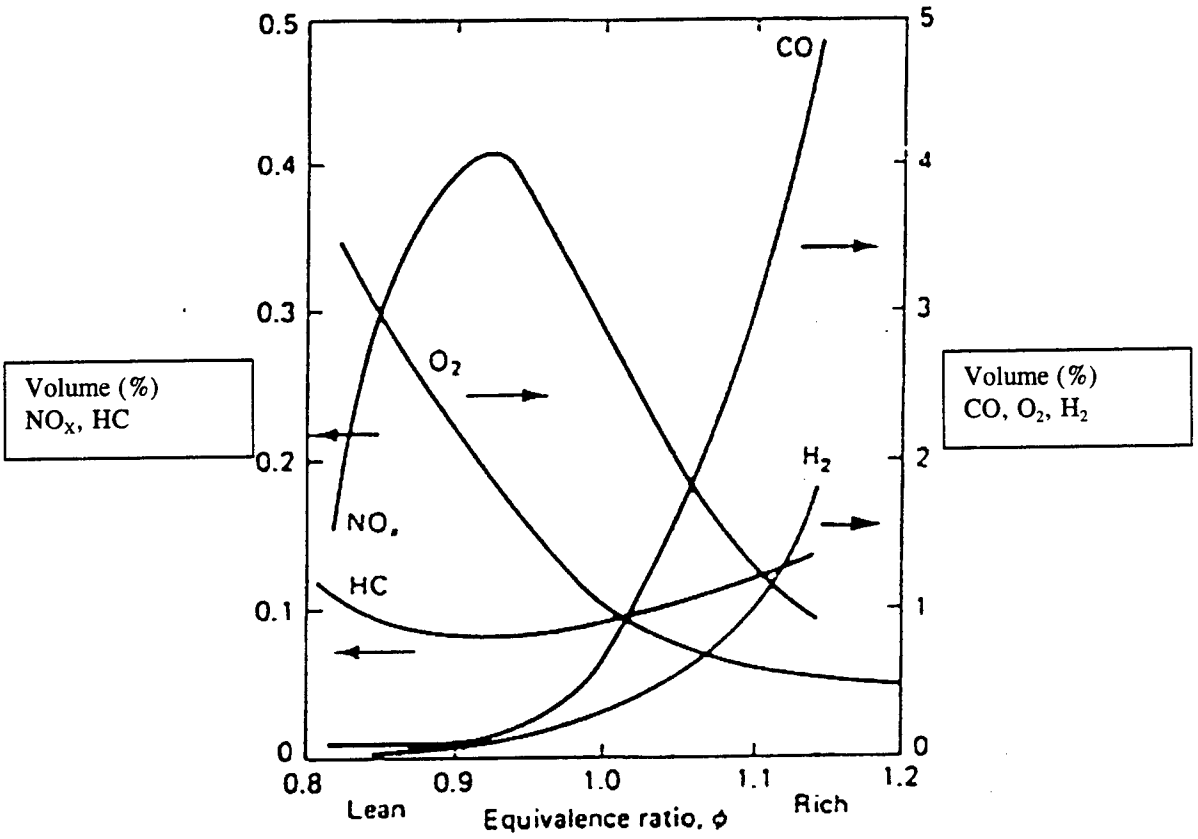



Figure 1.3 Exhuast gas concentrations [14]

By sensing the amount of gas present within the exhaust, it is possible to determine the approximate equivalence ratio at which the system is operating at any

given moment. Figure 1.3 illustrates a typical concentration of exhaust gases. As is evident from the bottom curve of Figure 1.1, the most efficient level of combustion occurs at a point slightly leaner than the ideal. This occurs because true complete combustion is impossible. Maximum combustion of the fuel present within the system is only obtained with excess oxygen. Regulating the fuel-to-air ratio of the system is crucial in order to control combustion efficiency.

# II SYSTEM CONSTRUCTION

The initial stage of the project consisted of manufacturing the experimental system. The complete system integrates the exhaust flue, burner assembly, fuel source, air source, sensor, sensor signal conditioner, and a personal computer (PC). In Figure 2.1 the actual configuration of the final version of the experimental system is shown. Appendix D describes the physical construction of the system.

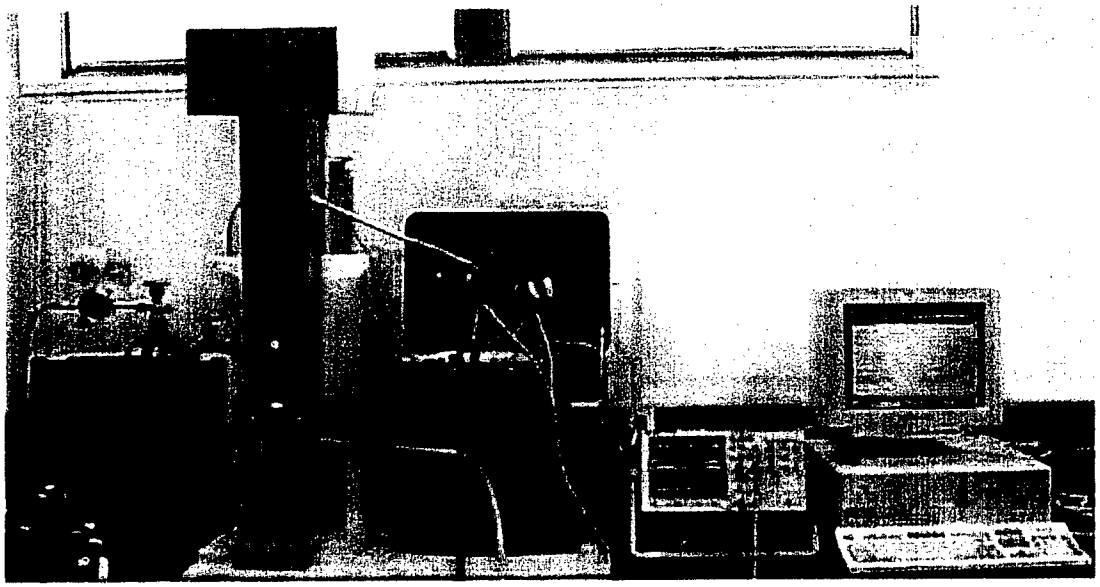


Figure 2.1 Experimental System

## SENSOR IMPLEMENTATION

### Sensors and Feedback

A sensor is included within the system design to provide feedback. The controller uses the difference between the desired operating point and the sensor output as a means of regulating the operation of the burner. This comparison between the desired operating point input into the system, and the actual point determined by the sensor, is called negative feedback. The error signal generated by this comparison is then fed into the controller in order to minimize the difference between the

system's actual and desired operating points, and thereby to force the system to operate at the desired level. A final advantage is that feedback makes the system relatively insensitive to external disturbances and small variations within the system itself.

A Meggitt Avionics type 624 sensor (Figure 2.2) provides feedback to the system by measuring the carbon monoxide concentration in the exhaust gases. The properties that make this sensor desirable are its resistance to high temperatures, its reasonably linear output across the system's range of operation, and its ability to sense oxygen, carbon dioxide, and carbon monoxide. The sensor is composed of a series of thermo-couples that absorb infra red (IR) radiation at various frequencies. When gases are heated, they emit IR radiation at specific frequencies. Consequently, when the thermo-couples are exposed to different levels of IR radiation from heated gas, their temperatures increase.

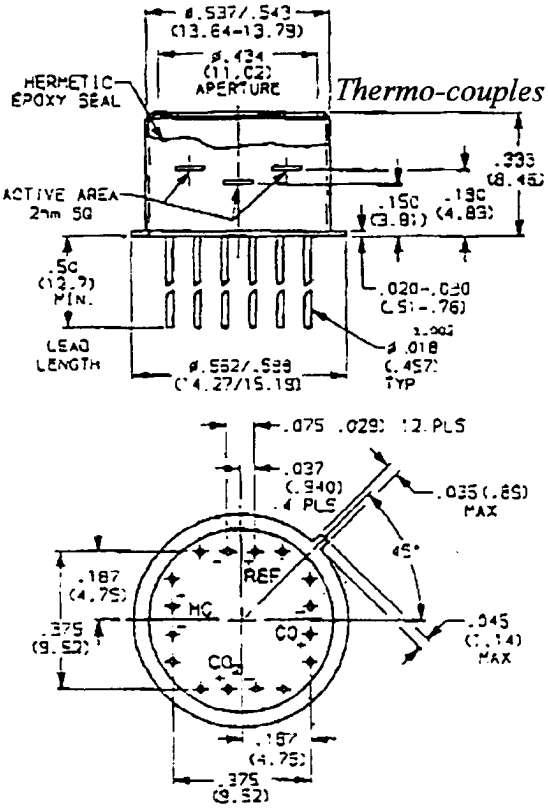



Figure 2.2 Gas Sensor

This difference in temperature, when compared to a reference thermo-couple, creates a voltage that corresponds to a specific gas concentration. However, the voltage output by the sensor is too small to be used by the controller, and is severely cluttered by high frequency noise. Signal conditioning is employed to solve both these problems.

## Sensor Protection and Signal Conditioning

The carbon monoxide sensor is currently fitted to the side of the exhaust flue prior to the first "S" curve. It rests within a 1/8 inch thick Teflon bushing 8 ½ inches above the base of the flue. The Teflon bushing acts to insolate the sensor from the

high temperatures the flue reaches during the operation of the system. If the sensor were not protected from the temperatures reached by the flue walls, the thermo-couples within the sensor would be affected by this additional heat source, and would output voltages corresponding to incorrect gas concentrations.

The carbon monoxide sensor signal is conditioned using a LM 324 operational amplifier, resistors, and capacitors. Since the signal output by the sensor is too weak and cluttered with noise to be used by the controller, the circuit constructed both filters and amplifies the signal.

The first stage of the signal conditioner is used to boost the sensor output to a level that can be easily read by the PC. This is accomplished by means of a non-inverting amplifier. The amplifier increases the voltage of the signal in a manner represented by equation (2.1).

$$\frac{V_{out}}{V_S} = 1 + \frac{R_F}{R_S} \tag{2.1}$$

The amplifier itself is depicted in Figure 2.3. $V_{out}$ is the voltage output by the amplifier, $V_S$ is the input voltage, $R_F$ is the resistance subjected to the feedback current, and $R_S$ is the resistance leading to ground. The non-inverting amplifier operates by using a very high input resistance, and very small output resistance. Simply by altering the resistance ratio between the feedback and ground resistors, the gain applied to the input signal can easily be changed. Consequently, to correct for the effects of the flues wall's temperature,a thermister with a thermal sensitivity roughly opposite that of the sensor was placed in the feedback path of the amplifier. The $R_F$ resistor in Figure 2.3 represents this location. The thermister itself rests within the Teflon bushing beside the sensor. As the temperature of the sensor increases, its voltage output also increases. When the thermister increases in temperature, its resistance decreases. In this manner, decreasing the gain of the amplifier compensates for the increased voltage output of the sensor.

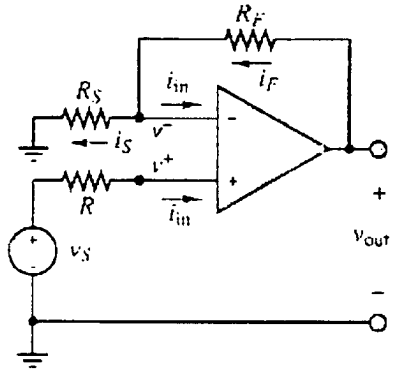



Figure 2.3 Non-inverting Amplifier [13]

Once the signal has been amplified, its inherent high frequency noise must be removed. This noise acts to mask the useful signal, and must be filtered out in order to provide the controller with usable data. A low pass filter, depicted in Figure 2.4, is used to remove the high frequency component of the signal. The voltage output by the filter contains only the low frequency components of the sensor signal, and is usable by the controller.

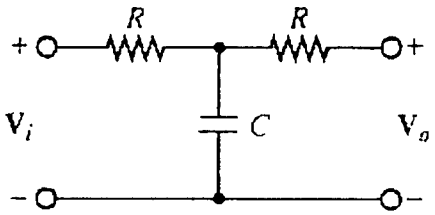



Figure 2.4 Low Pass Filter [13]

Although equations do exist that represent the operational characteristics of the filter type used, it was necessary to experimentally determine the capacitance and resistance values that provided the optimum performance. Neither the frequency of the desired output, nor the frequency of the noise were initially known.

The final stage of the signal conditioner is the inverting amplifier. This component functions to invert the signal to a positive value. The voltage initially generated by the sensor is negative, and is inverted only as a matter of convention. Figure 2.5 depicts an inverting amplifier, and equation 2.2 illustrates its behavior.

$$\frac{V_{out}}{V_S} = -\frac{R_F}{R_S} \qquad (2.2)$$

$V_{out}$ is the output voltage, $V_S$ is the input voltage, $R_F$ is the feedback resistance, and $R_S$ is the negative input resistance. Although no amplification is used during this stage, (both resistors possess equal values) the voltage of the signal may be easily boosted if it should become desirable. Allowing for flexibility in the design has proven useful within the project. Figure 2.6 depicts the entire signal conditioning circuit as it was actually implemented.

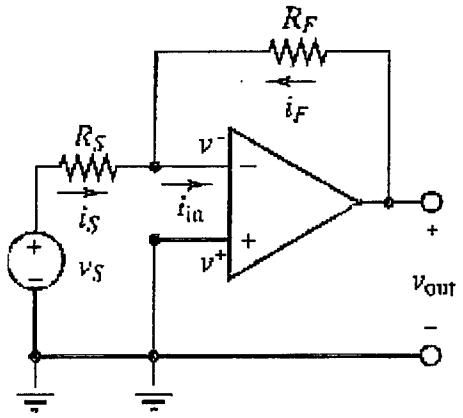



Figure 2.5 Inverting Amplifier [13]

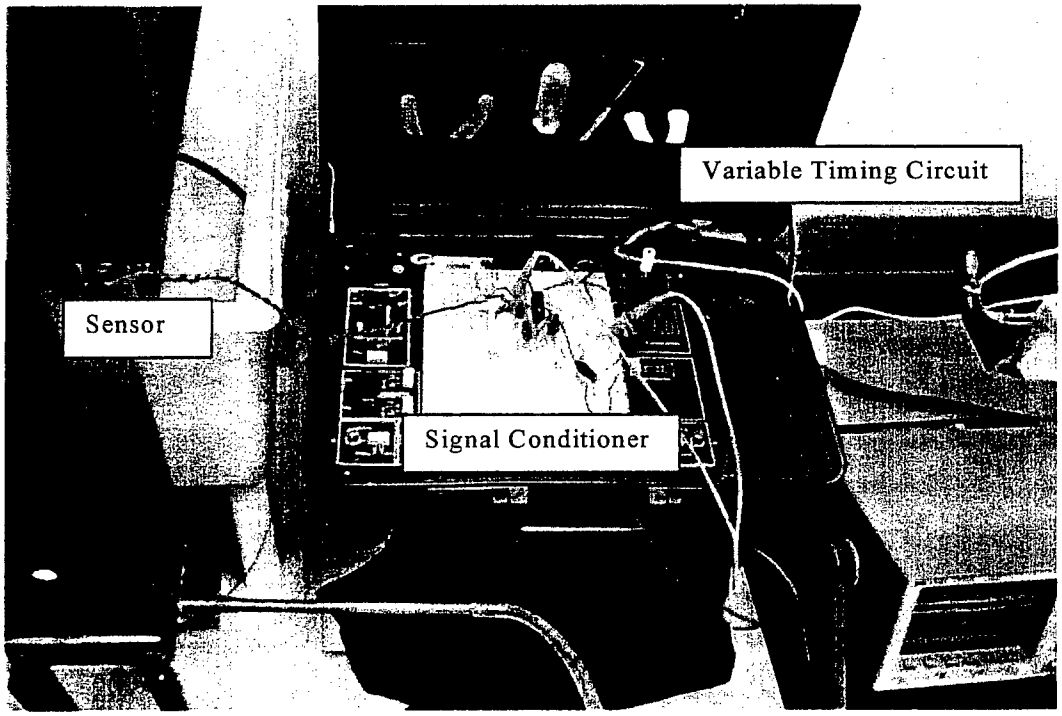



Figure 2.6 Signal Conditioning Circuit

## Sensor Calibration

Before the sensor will generate useful information, it must first be calibrated so that the output voltage will correspond to a specific operating point. (Although combustion is dependent upon many factors, including the mixture of air and fuel in the burner, the model-based approach studied within this project will correct for many of these variables. For a given amount of air, the fuel input to the system will be regulated to achieve the desired exhaust output.) From existing literature, it was determined that the most efficient combustion of methane (the point closest to complete combustion that is actually possible) occurs with approximately 3% excess oxygen within the exhaust gases [5]. By varying the fuel flow into the burner, and observing the oxygen content using a portable gas analyzer, it was determined that the correct sensor output for this operating point (3% excess oxygen) was .5 volts. Although the sensor itself detects CO levels within the exhaust gases of the burner, there is a unique CO and $O_2$ concentration that corresponds to the fuel-to-air ratio at which the system is operating. Therefore, throughout the system's range of operating temperatures and concentrations of exhaust gases, it is possible to calibrate the CO sensor by detecting the $O_2$ content of the exhaust gases. This characteristic of combustion is evident within Figure 1.3 where the CO and $O_2$ concentrations are plotted as curves against a generalized equivalence ratio for combustion.

## Computer Interface

The control algorithms for the system are computer programs written in C++. The personal computer (PC) is a digital computer. This means that it operates through a series of discrete voltages pulsed at discrete time intervals. However, the sensor, signal conditioner, and solenoid valves are analog devices. They operate by continually transmitting, as in the case of the sensor and signal conditioner, or receiving, as in the case of the solenoid valves, non-discrete voltages. To allow these devices to be compatible, an Analog to Digital/Digital to Analog (AD/DA) converter must be used.

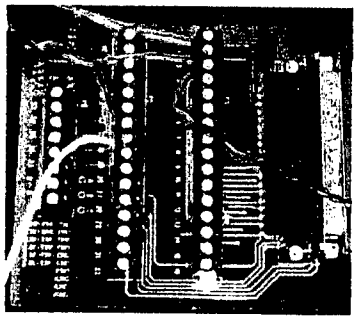


Figure 2.7 AD/DA Converter

Shown in Figure 2.7, a CIO-DAS08/Jr-A0 converter manufactured by Computer Boards, Inc. is used to interface the computer with the analog devices of the system. The converter operates by accepting the analog signal from the signal conditioner, and digitizing it at an adjustable sampling rate set to 0.1 seconds. The PC then manipulates the digital signal and outputs the result back to the converter. The digital signal from the PC is converted to an analog voltage that is then used to drive the solenoid valve regulating the fuel feed. The converter possesses eight analog to digital channels, and two digital to analog channels. This enables the AD/DA converter to simultaneously input digitized data to the PC, and output analog voltages to the solenoid valve.

The sampling rate of the AD/DA converter is set to 0.1 seconds to allow the PC enough time to complete one loop of the control program after each digital sample is taken. This is accomplished by using a timing circuit that generates a square wave with a period of .01 seconds. The value of the square wave changes every 0.005 seconds, and a C++ program loop initiates each new sample after every 20 voltage changes. Although this sampling period is large in comparison to most other digital applications, it is still significantly smaller than the time constant of the system. This ensures that the sampling rate is still fast enough to allow system to be controlled effectively.

# III PROPORTIONAL INTEGRAL (PI) CONTROL DESIGN

## PI CONTROL OVERVIEW

The first controller designed and implemented within the system was of the proportional integral (PI) type. This control scheme is commonly used in many industrial applications. The primary advantage of the PI controller is that a model of the plant is not required to control the desired system. Provided that the requirements are not stringent, the inherent flexibility of this control design makes it suitable for most simple control problems. Equation 3.1 depicts the PI control algorithm.

$$u(t) = K_c\left( e(t) + \frac{1}{T_i}\int_0^t e(s)ds \right)$$

where

$$e = u - y \tag{3.1}$$

The variable $u$ is the control input, $e$ is the error signal, and $y$ is the process output. $K_c$ is the proportional gain and $T_i$ is the integral time. These last two variables are regulator parameters that are chosen to optimize the performance of the controller.

Since the PI controller is not based upon a model of the system that it seeks to control, its accuracy is limited. PI control becomes increasingly ineffective as the complexity of the system increases. In fact, since PI control increases the type (the number of integrations indicated by the open loop transfer function) of the compensated system by 1, it causes the system to become less stable, or even unstable.[10] To avoid instability, selection of the $K_c$ and $T_i$ values must be conducted with care. Although, through cautious design procedures, system overshoot to a desired value may be eliminated entirely, the speed of the response will slow. This happens because the PI controller is a low pass filter that attenuates the high-frequency components of the signal.[10]

## PI CONTROL IMPORTANCE

The PI controller was implemented during the project to provide a basis for comparing the success of the final control designs. As mentioned in the previous section, PI and related controllers represent the standard means of control used within industry and most other applications requiring the control of combustion. The design and implementation of the PI controller also provided an important opportunity to test the experimental system. This allowed problems and potential difficulties to be identified and corrected before more complex control programs were initiated.

## ZIEGLER-NICHOLS TUNING OF THE PI CONTROLLER

The two Ziegler-Nichols approaches for tuning PI, and related controllers, provide an approximate means of determining the proportional gain ($K_c$) and integral time ($T_i$) necessary for the efficient operation of the controller. These methods are both conducted through experiments upon the plant, and are based upon the transient response characteristics of the plant. Ziegler-Nichols tuning methods provide only a basis for parameter values. Both methods seek to achieve a maximum overshoot of 25%. They are a starting point for further fine tuning of the controller [10].

### Method 1

The first approach requires an experimental determination of the plant's response to a step input. An S-shaped curve will be generated if the plant does not contain any dominant complex-conjugate poles. Figure 3.1 depicts an example of such a curve. A plot of the plant's response to the step input is characterized by the time constant $T$ and the time delay $L$. The time constant is the time the system takes to achieve 63% of its steady state value. The time delay is determined by drawing a line tangent to the inflection point of the S curve, and locating the point where it intersects the time axis. $K_c$ and $T_i$ can be determined using equations 3.2 and 3.3.

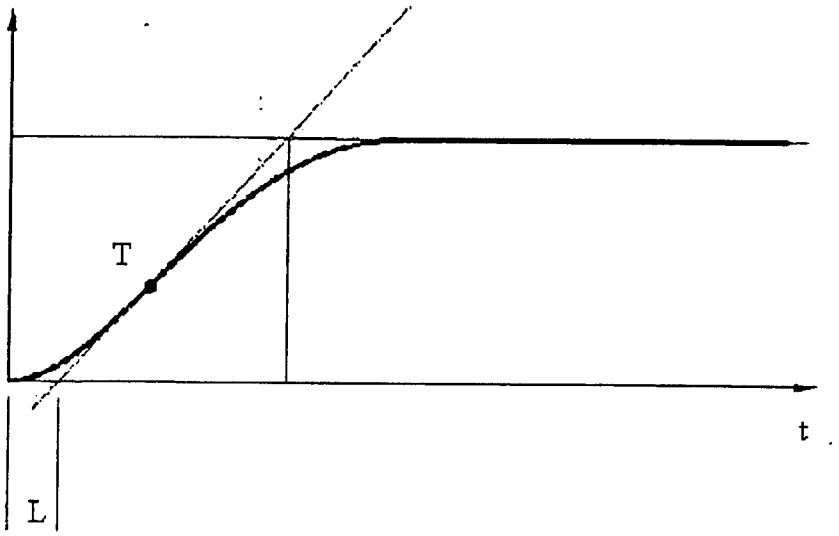



Figure 3.1 System Step Response

$$T_i = \frac{L}{.3} \tag{3.2}$$

$$K_c = .9\frac{T}{L} \tag{3.3}$$

Transforming equation 3.1 into Laplace form and substituting for $K_c$ and $T_i$, the PI control algorithm appears as:

$$G_c(s) = \frac{.9\,Ts + .3}{Ls} \tag{3.4}$$

## Method 2

In the second Ziegler-Nichols method, the integral time $T_i$ is set to an infinitely large value and the proportional gain $K_c$ is initially set to zero. The

proportional gain is then slowly increased until the plant exhibits sustained oscillations. This value of the proportional gain is called the critical gain $K_{cr}$. The period of each oscillation is measured and recorded as the critical period $P_{cr}$. Figure 3.2 depicts an example of the desired output. $K_c$ and $T_i$ can then be determined using equations 3.5 and 3.6.

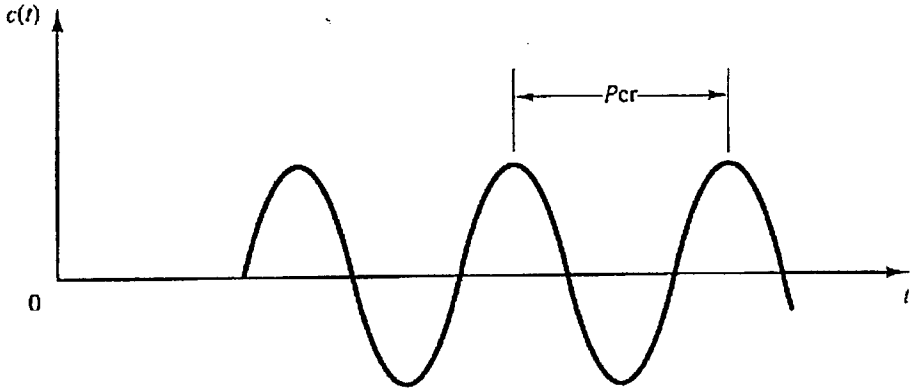



Figure 3.2 Sustained oscillation of system denoting critical period ($P_{cr}$)

$$K_c = .45K_{cr} \tag{3.5}$$

$$T_i = \frac{1}{1.2}P_{cr} \tag{3.6}$$

Transforming equation 3.1 into Laplace form and substituting for $K_c$ and $T_i$, the algorithm appears as:

$$G_c(s) = \frac{.45(K_{cr}P_{cr})s + 1.2}{P_{cr}s} \tag{3.7}$$

## Ziegler-Nichols Results

The first method of the Ziegler-Nichols tuning approach yielded the most favorable results. Graphing the system's response to a step input proved far easier than adjusting the controller's proportional gain to obtain a sustained oscillation. The time constant of the system is so large that oscillations in the output are not easily detected. Even when oscillations finally become noticeable, the system will starve the burner of fuel on the voltage minimums of each cycle and the flame is extinguished.

## PI CONTROLLER IMPLEMENTAION

PI control may be implemented using either analog electrical components, or a PC equipped with an AD/DA converter. Although analog components are easier to work with when only PI control is desired, digitizing the PI design enabled the model-based controller to be implemented later with only minor modifications to the original PI control program. In addition, once the difficulties of interfacing the PC with the system's analog components were solved, changing the proportional gain and integral time of the PI controller only required these values to be changed within a program. An equivalent analog PI controller would require the resistance and the capacitance of the controller to be changed by adding and removing electrical components.

The PI control program was written in C++, and is included in Appendix B. The program operates by importing values from the AD/DA converter, manipulating these data in a manner directed by the PI controller, and exporting a voltage through the AD/DA converter that then drives the valve regulating the fuel flow to the burner. However, because the PC is a digital system, the PI controller must be converted into a form compatible with the discrete sampling period. This derivation is included in Appendix E.

## PI CONTROL RESULTS

Once the nomimal values for $K_c$ and $T_i$ were determined, it was necessary to fine tune the PI controller through a series of experiments. This procedure was nothing more than using trial and error to develop the best control parameters. In

addition, the system itself imposed limitations on the magnitude of the voltage that could be generated without causing damage to the AD/DA converter. Voltages greater than five volts can not be processed by the converter. It was also necessary to add a segment to the control program which would not allow the control effort to drop below 0.5 volts. Without this addition, the burner is extinguished during each trial because the flame can not be maintained with a voltage of less than .5 volts being sent to the solenoid valve.

The following series of plots depicts the system's operation under varying control parameters. The jagged nature of the plots results from sensor noise that could not be entirely removed by the low pass filter. As the gain of the system was increased with each trial, the noise was amplified by the controller, and becomes increasingly evident in the control effort.

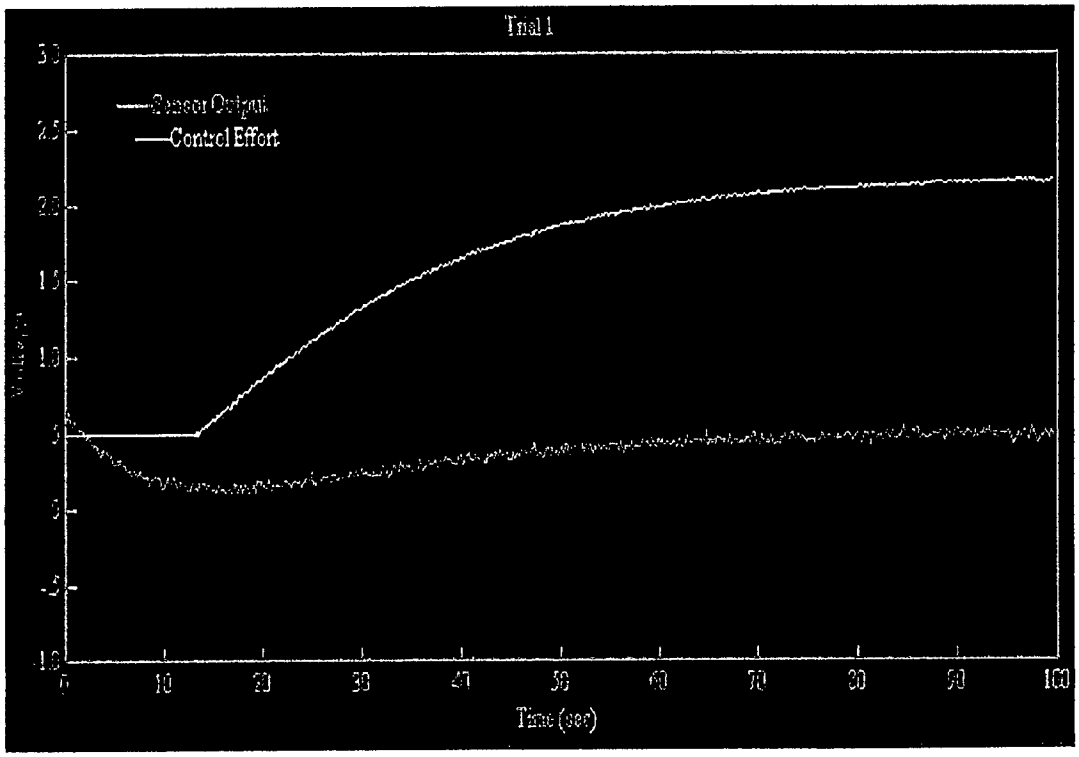



Figure 3.3 PI Controller with initial $K_c$ and $T_i$ paramteters

Figure 3.3 depicts the plots of the sensor output and control effort for the first trial values of $K_c$=.2588 (controller gain) and $T_i$=1.667 (integral time). Each simulation was begun at an arbitrary sensor value of 0.6 volts, and each controller was tasked with operating the system at a value of 0.5 volts. Once again, 0.5 volts

corresponds to the carbon monoxide content at which roughly 3% excess oxygen is also present. This represents the most efficient level of combustion. As noted in the section entitled "Sensor Calibration," the ideal operating point for the system was selected to correspond to a sensor output of 0.5 volts.

The first trial illustrates that the controller possesses reasonable characteristics. The control effort peaks at 2.45 volts, and the sensor output eventually settles to a value of 0.5 volts. However, the response time of the system is rather slow. In order to improve this aspect of the system, the overall gain of the controller was adjusted over a series of trials to find the one that generated the best results. Figures 3.4, 3.5, and 3.6 depict the PI control results as the gain was modified.

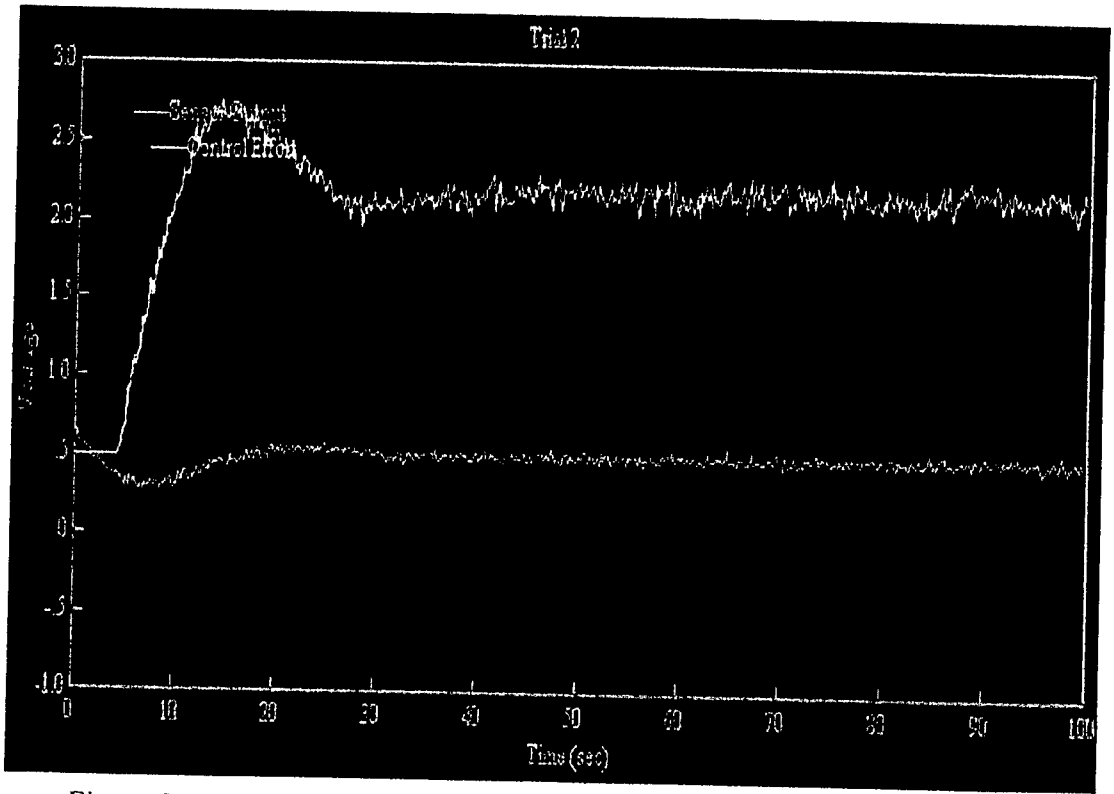



Figure 3.4 PI Controller with $K_c$ and $T_i$ paramteters increased by a factor of 10

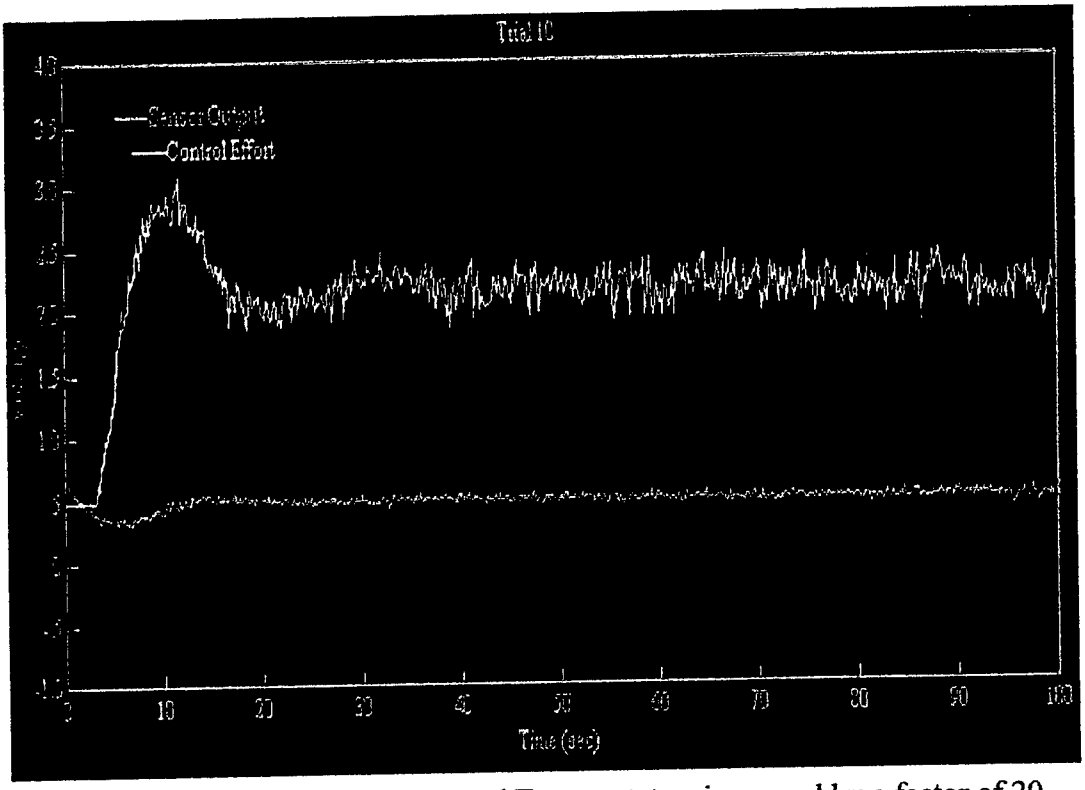



Figure 3.5 Controller with $K_c$ and $T_i$ paramteters increased by a factor of 20

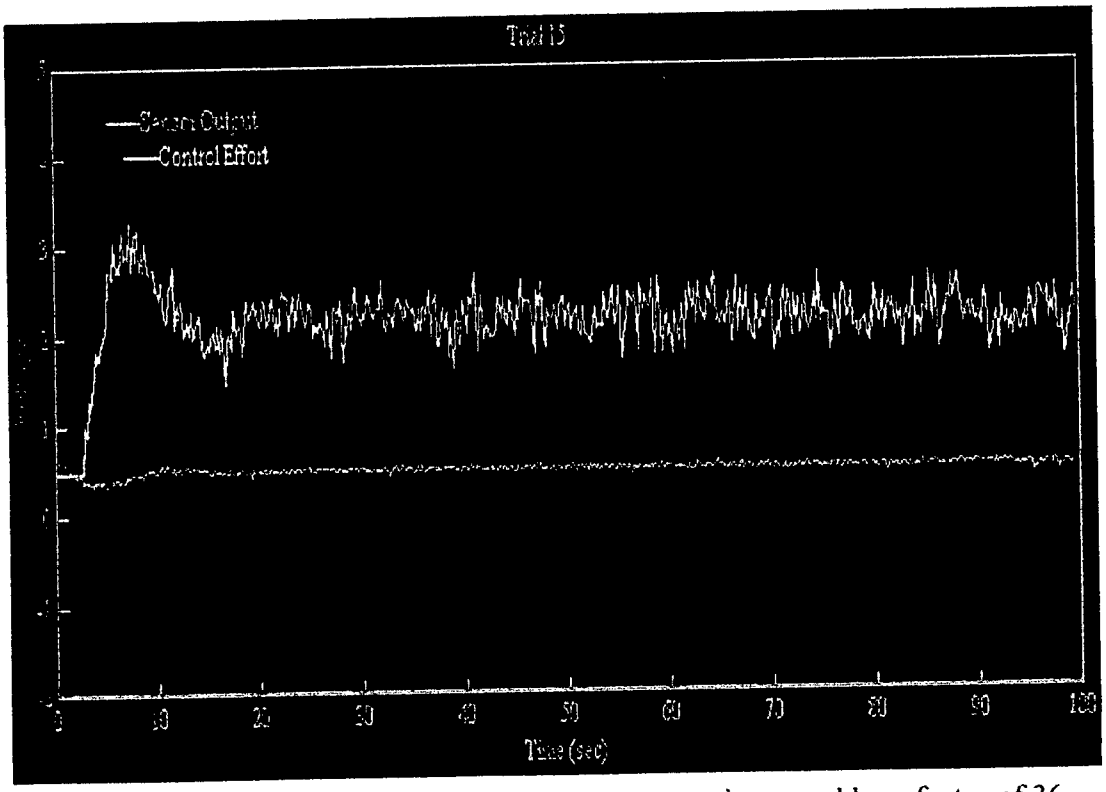



Figure 3.6 PI Controller with $K_c$ and $T_i$ paramteters increased by a factor of 36

Each of the above trials depicts an improved response time as the overall controller gain was increased. The final PI control design was derived through experiment fifteen, which is depicted in Figure 3.6. The gain can not be increased above 36 or the control effort will peak above five volts, and damage the AD/DA converter. Also, because of the minimum voltage restriction set upon the control effort, even if the gain could be increased beyond 36, the response time would not be significantly affected. The solenoid valve is never given a voltage below 0.5 volts, and therefore each system requiring a theoretical voltage below this level will respond in a similar manner. This point is evident upon a comparison of the responses pictured in Figures 3.5 and 3.6. Both trials achieve a steady state value within approximately ten seconds, although Figure 3.6 is slightly faster.

# IV MODEL-BASED DESIGN

## MODEL IDENTIFICATION

The most important step in the creation of a model-based control design is the construction of an accurate model of the system to be controlled. The system parameters must be correctly estimated or the resulting control designs will not operate properly. The basis for the identification of the experimental system within the project was a procedure known as recursive least squares estimation.[2] This method was used to produce linear models of three different operating points. In a manner that will be discussed in the next section, a non-linear control scheme was effected from these linear models.

The principle of least squares was first formulated by Karl Gauss near the end of the eighteenth century, and was used to estimate the orbits of planets and asteroids. The procedure seeks to estimate the unknown parameters of a system such that the sum of the squares of the differences between the actual and the computed values, multiplied by numbers that measure the degree of precision, is minimized.[2] The resulting equations for recursive least squares estimation appear as the following:

$$\begin{gathered}
\hat{\theta}(t) = \hat{\theta}(t-1) + K(t)(y(t) - \varphi^T(t)\hat{\theta}(t-1)) \\
K(t) = P(t)\varphi(t) \\
P(t) = P(t-1) - P(t-1)\varphi(t)(I + \varphi^T(t)P(t-1)\varphi(t))^{-1}\varphi^T(t)P(t-1) \\
P(t_o) = (\Phi^T(t_o)\Phi(t_o))^{-1} \\
\Phi^T(t_o)\Phi(t_o) = \sum_{i=1}^{t} \varphi(i)\varphi^T(i)
\end{gathered} \tag{4.1}$$

where the model appears in the form:

$$y(i) = \varphi_1(i)\theta_1^0 + \ldots + \varphi_n(i)\theta_n^0 \tag{4.2}$$

$\theta$ represents the parameters to be determined, $\varphi$ represents the known functions, and y is the output of the model to be determined. $\varphi$ and $\theta$ are treated as vectors in the

equations of 4.1. The number of terms the model contains is variable, and must be chosen before the process is begun.

## IMPLEMENTATION

C++ was employed to implement recursive least squares estimation for the combustion system, and the program is included in Appendix A. A program was written that generated a random input to the solenoid valve, between .5 and 5 volts, and sampled the sensor output every 0.1 seconds. The random input to the solenoid valve was necessary to derive the dynamic characteristics of the system.

Before the program was implemented within the combustion system, it was tested using VisSim. An arbitrary transfer function was chosen, and a random input was fed into the simulated system. The simulated system's inputs and outputs were saved into files, and were fed into the C++ program to verify its correct operation. Once the program was debugged, it consistently returned the parameters of the original transfer function. For example, when the transfer function:

$$G(z) = \frac{z+2}{z^3 - .8z^2 + 1.5z + 0} = \frac{b_1 z + b_0}{z^3 + a_2 z^2 + a_3 z + a_4} \tag{4.3}$$

was simulated, and its inputs and ouputs were given to the C++ program, Figures 4.1 through 4.5 were generated.

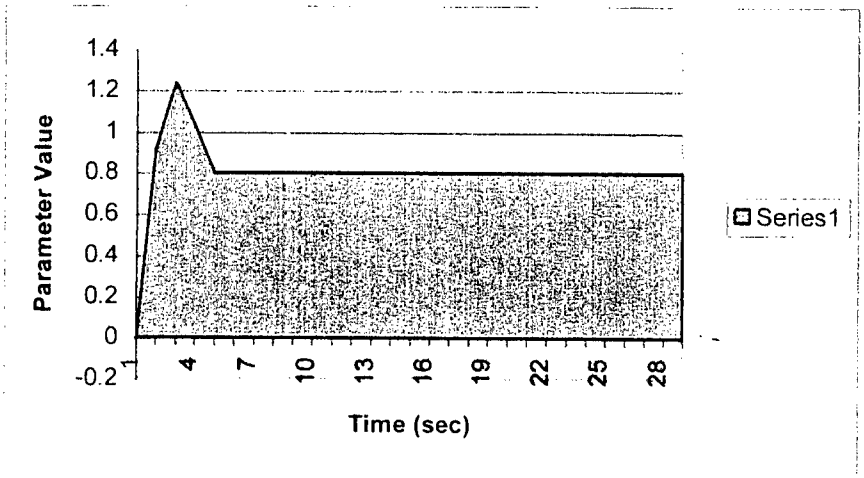



Figure 4.1 $a_2$ settling to the model parameter of 0.8

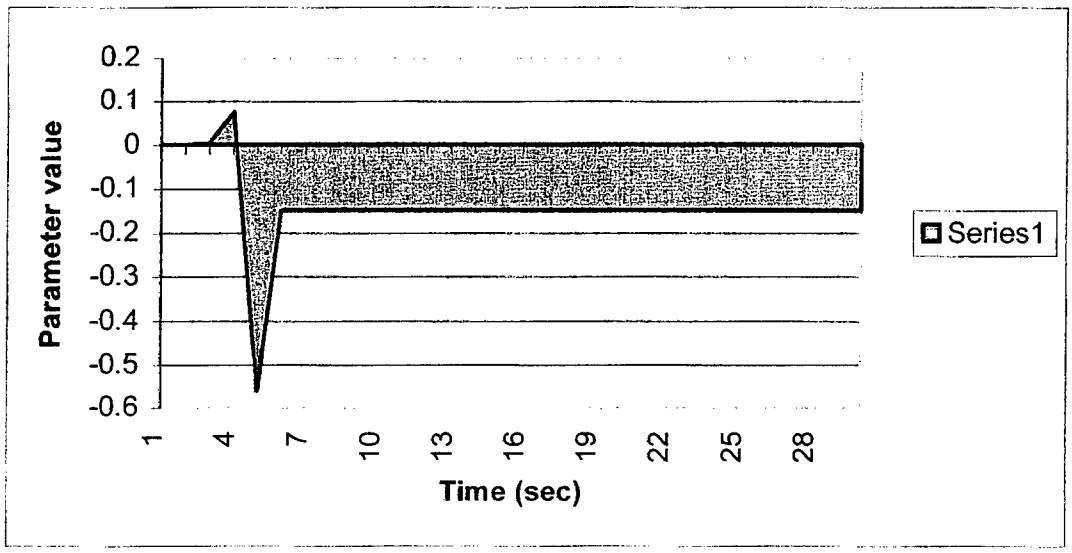



Figure 4.2 $a_3$ settling to the model parameter of -1.5

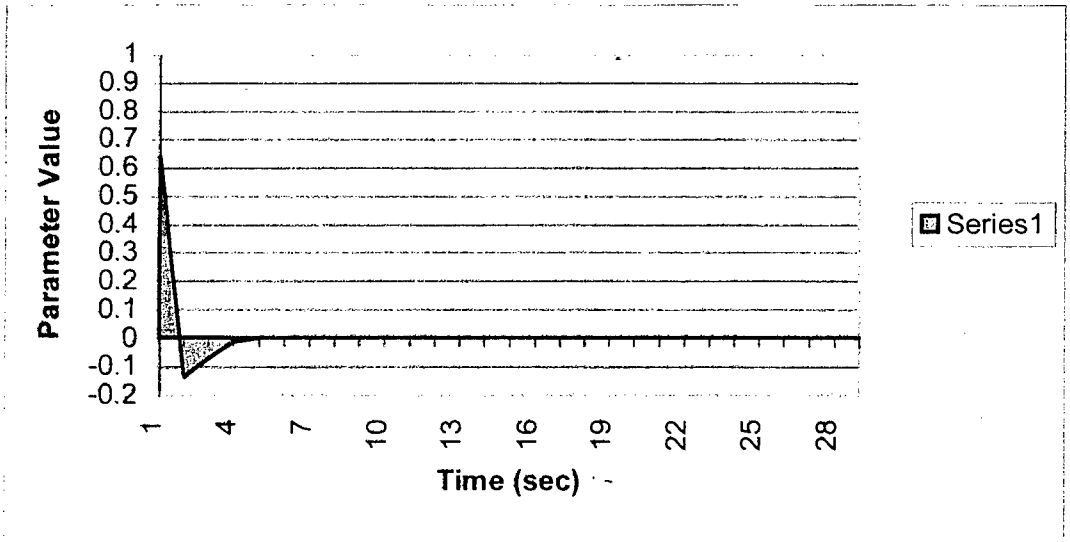



Figure 4.3 $a_4$ settling to the model parameter of near zero

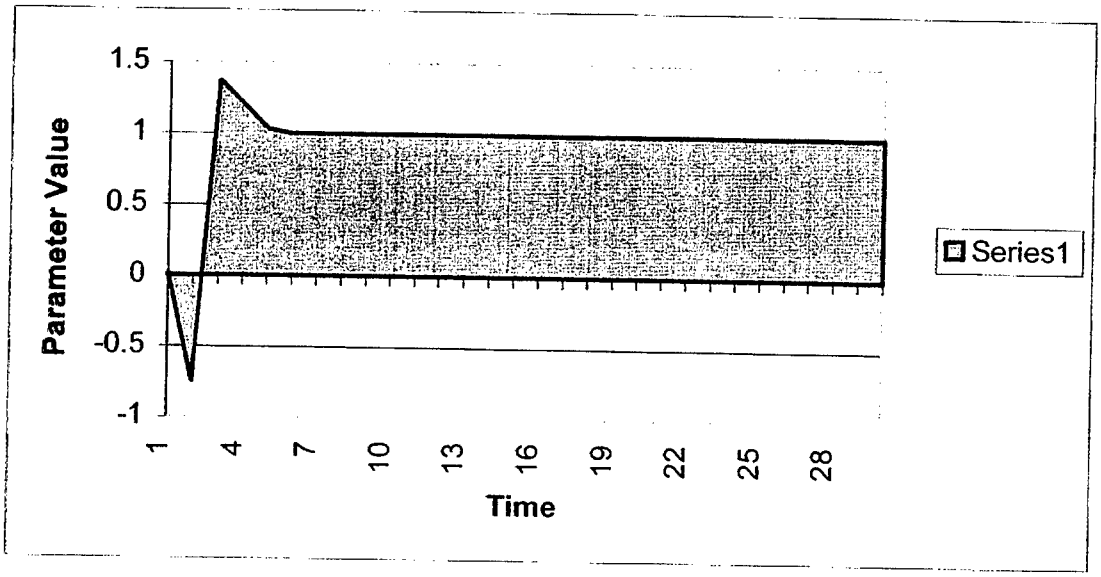



Figure 4.4 $b_1$ settling to the model parameter of 1

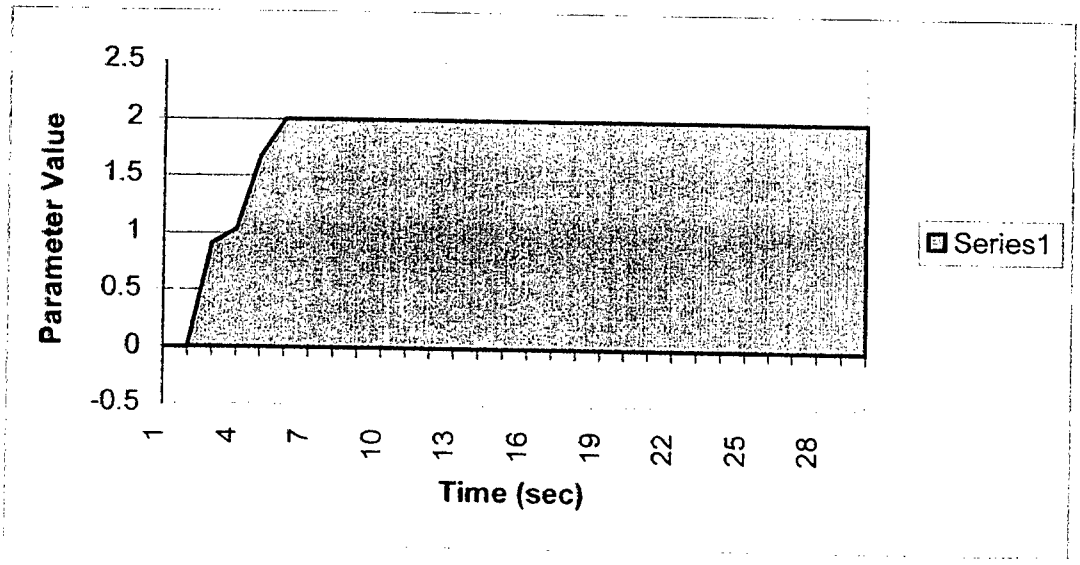



Figure 4.5 $b_0$ settling to the model parameter of 2

The initial model structures are always chosen in a manner such that the first coefficient of the denominator is one. The program output is given in a form which is readily converted into a difference equation, and the one in the denominator simplifies this process. Similarly, the denominator outputs from the program are the inverses of those found in the original transfer function. Once again, this simply aids in the conversion of the program's output into a difference equation that can be later used in the control program. Bearing in mind the information that was discussed above, the program outputs are nearly identical to the original parameters of the transfer function in equation 4.3.

Once the debugging and testing of the program was completed, it was run with the experimental system. The initial plan called for the system to be operated under three different loading conditions. The first was with no water in the cooling tanks, the second with the first cooling tank filled, and the third with both tanks filled. The program was used to derive models for each of these conditions. However, even with each tank filled to its maximum capacity, the loading conditions failed to effect any considerable change from the models derived under no loading conditions.

A solution was found by varying the air introduced to the burner. By regulating the air input, loading conditions were emulated by operating the system under lean, near stoichiometric, and rich conditions. The lean condition was satisfied by introducing more air than was required for the combustion of the fuel, the near stoichiometric point was satisfied by allowing roughly 3% excess oxygen to be expelled in the exhaust gases, and the rich condition was fullfilled by operating the system with less air than was required to completely burn the fuel. (The lean condition was characterized by roughly 6% excess oxygen, and the rich by roughly 0%-1% excess oxygen.) The models derived from each of these conditions are distinct.

Using VisSim, the actual operation of the system was compared with the simulated operation of the model. Model structures with varying numbers of poles and zeroes were experimented with until the simulated operation of the model was similar to the actual output of the system. The plots depicted below are the results of the models selected for each of the three operating points. The models selected are given below as transfer functions.

$$G(z) = 0.00172242\frac{z+.849}{z^5-.961226z^4-.0209421z^3-.00112333z^2-.0174243z+.0146176}$$

(4.4: Near Stoichiometric Model)

$$G(z) = 0.00047797\frac{z+.019174}{z^5-.955412^4-..0246399z^3-.00105189z^2-.00338509z-.0138404}$$

(4.5: Lean Model)

$$G(z) = 0.000969177\frac{z-.7298}{z^5-.993659^4+.0105317z^3+.00546735z^2-.0183735z-.0321175}$$

(4.6: Rich Model)

A model structure possessing five poles and one zero was found to approximate the system best at each operating point. When fewer parameters were used in the model structure, a good fit to the actual system data could not be attained. Conversely, when more parameters were incorporated into the model structure, a good fit for the particular experiment could be obtained, but the model could not produce favorable results for another experiment conducted under similar conditions. (This good fit to the measured data, but poor fit to a similar data set is called overfitting [2].)

Figures 4.6, 4.7, and 4.8 are comparisons between the actual plant output (in yellow), and the simulated plant output (in blue) using the models derived for each of the system's operating points. It is apparent from these figures that each model is successful in approximating the system's performance when subjected to similar conditions.

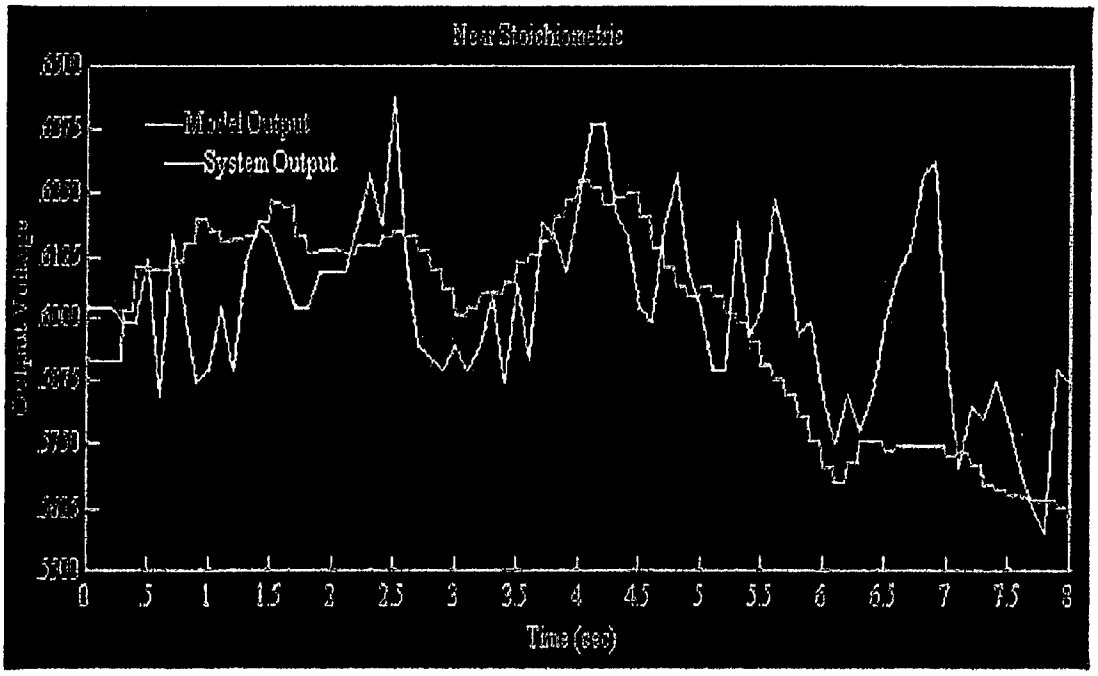



Figure 4.6 Comparison of model and system operation for near stoichiometric conditions

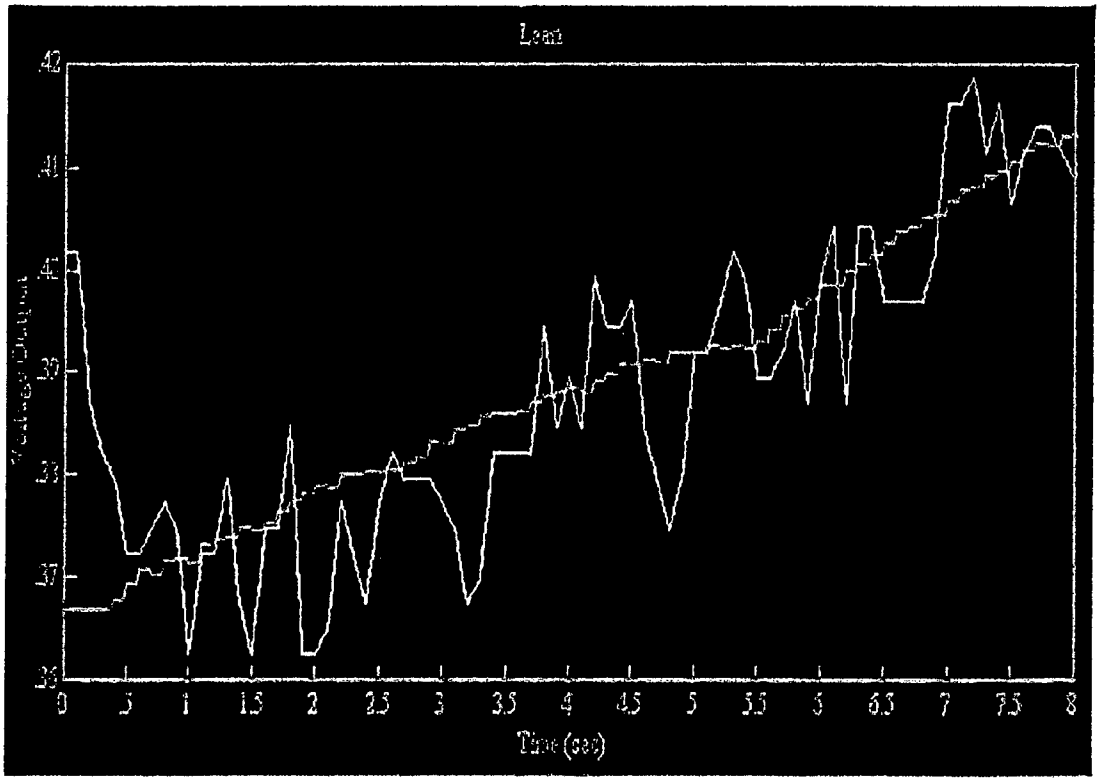
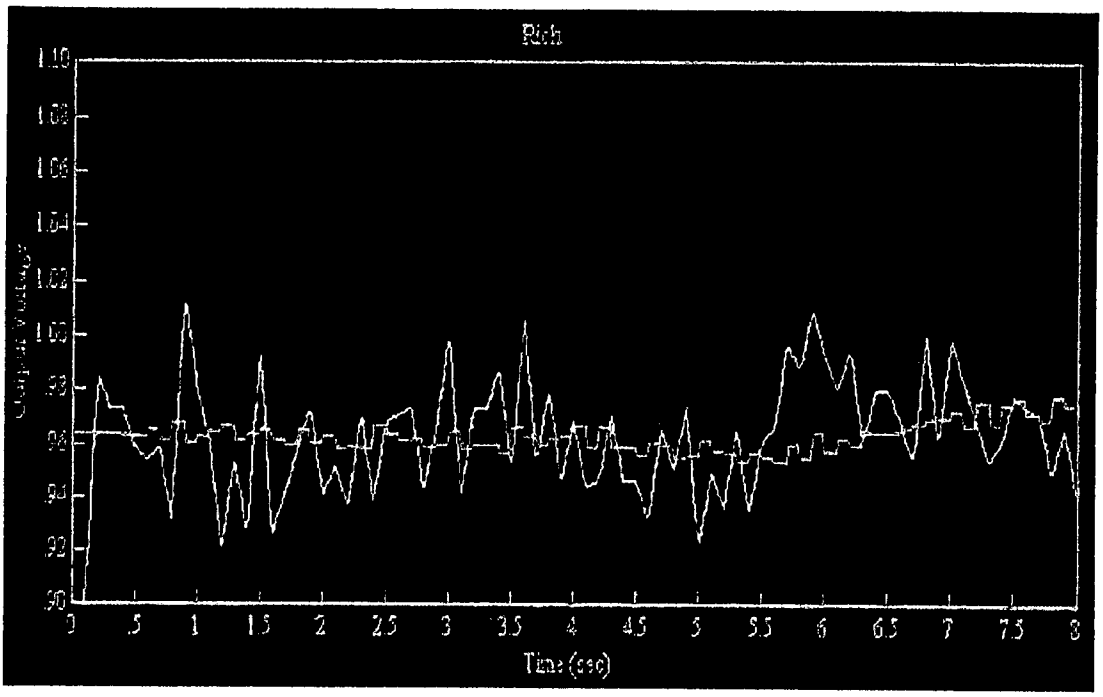



Figure 4.7 Comparison of model and system operation for lean conditions




Figuure 4.8 Comparison of model and system operation for rich conditions

# V MODEL-BASED CONTROLLER DESIGN AND IMPLEMENTATION

## DESIGN

Although a variety of control designs could have been used, the deadbeat control scheme was chosen to be implemented within the project. This design possesses the advantages of easy formulation, decreased response times, and smaller overshoots than PI and related controllers. The deadbeat strategy is also unique to discrete time systems, and its only design parameter is the controller's sampling period.

The control scheme attempts to shift the system's zeros to the origin of the z-plane, and will consequently drive the error to zero in, at most, the number of sampling periods corresponding to the order of the denominator of the closed loop system's transfer function. For example, a system possessing a fifth order transfer function and sampling time of one second will be have an error signal of zero after five seconds of operation. The design procedures for deadbeat control are given in the following equations.

$$K = \frac{1}{1+N}$$

$$D(z) = \frac{K(z+N)}{G(z)(z^5 - Kz - KN)} \tag{5.1}$$

D(z) is the transfer function of the controller, N is the plant zero, and G(z) is the transfer function of the plant. The transfer functions for each of the controllers are listed below.

$$D(z) = 85\frac{z^5 - .961226z^4 - .0209421z^3 - .00112333z^2 - .0174243z + .0146176}{z^5 - .5408z + .4591}$$

(5.2 Near Stoichiometric Control)

$$D(z) = 45\frac{z^5 - .955412^4 - ..0246399z^3 - .00105189z^2 - .00338509z - .0138404}{z^5 - .981187z - .0188}$$

(5.3 Lean Control)

$$D(z) = 45\frac{z^5 - .993659^4 + .0105317z^3 + .00546735z^2 - .0183735z - .0321175}{z^5 - .5781z - .4219}$$

(5.4 Rich Control)

## IMPLEMENTATION / RESULTS

The controllers were implemented using C++, and were simply inserted into the program structure that was created for the original PI controller. This combination of control functions allows the program to operate non-linearly and enables the system to be controlled over its entire operating envelope. The control concept is illustrated in Figure 5.1. The only experimental modification necessary to the control scheme was to decrease the gains of each of the separate controllers. This brought the control effort within the safety margins of the AD/DA converter, and decreased the system's oscillation about the desired operating point.

To operate the control program, the desired voltage (which corresponds to the desired equivalence ratio) to which the system will be driven is entered. When the program is initiated, it samples the sensor voltage, and decides which particular controller to use depending upon the instantaneous operating point of the system. In this manner, the program will automatically switch controllers when the system is driven into a new region. The boundary voltages used by the program are >0.6 volts for rich operation, <0.4 volts for lean operation, and all other values for near

stoichiometric operation. Figures 5.2, 5.3, and 5.4 depict the final control scheme along with the PI controller.

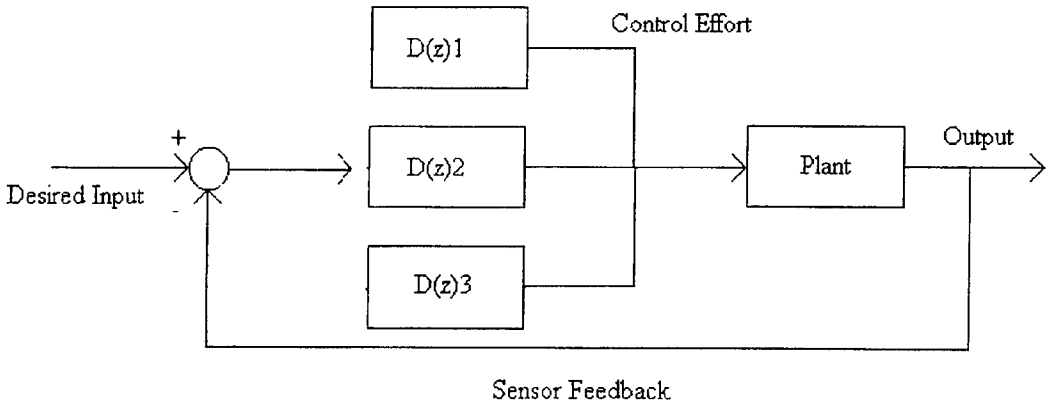



Figure 5.1 Model-Based Control Concept: Each control block D(z) represents a seperate model-based controller for the operating points chosen

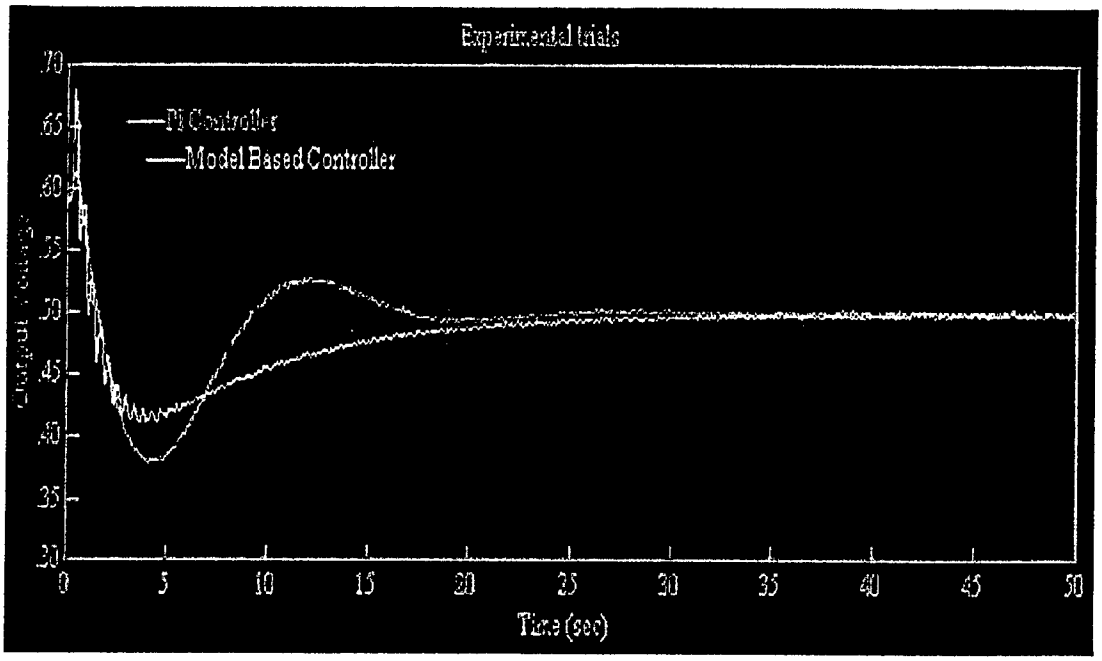



Figure 5.2 Comparison of system control to near stoichiometric operating point

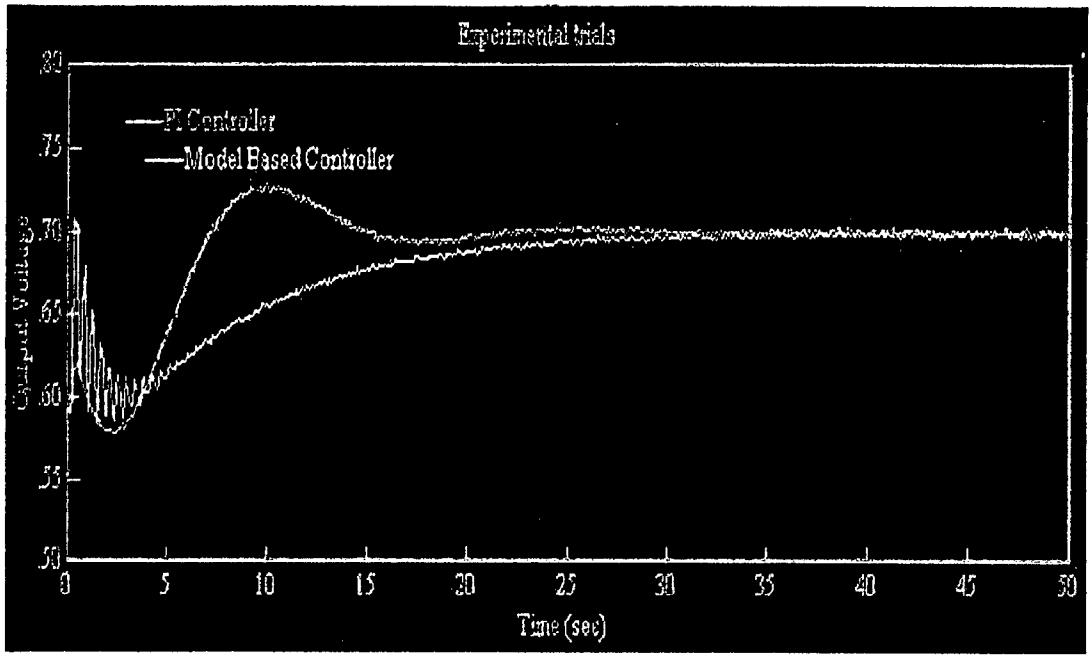



Figure 5.3 Comparison of system control to rich operating level

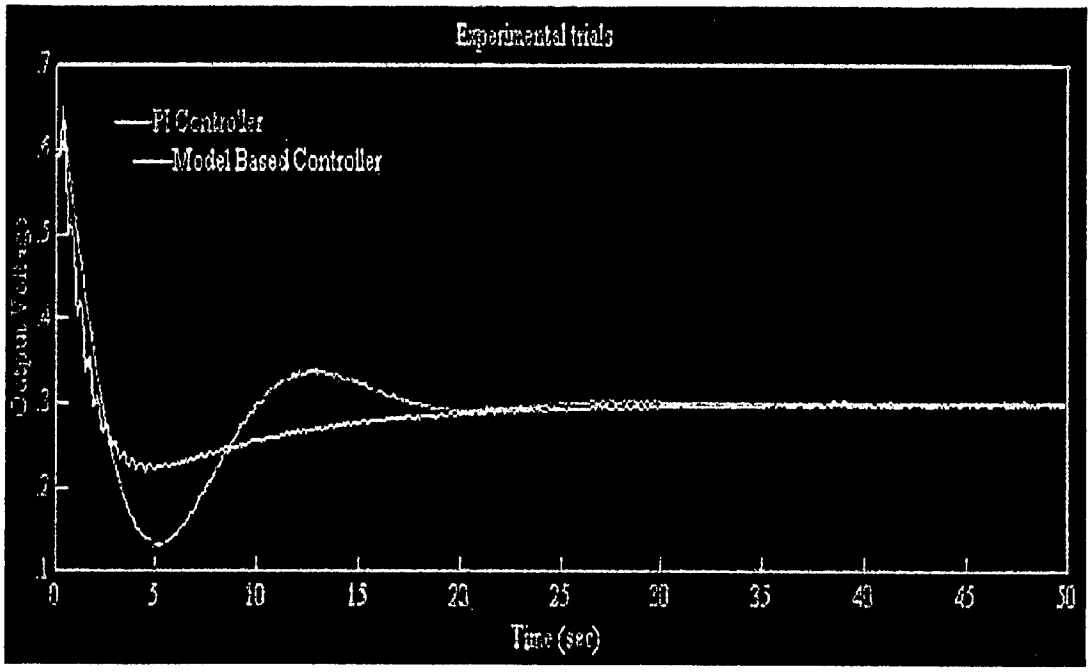



Figure 5.4 Comparison of system to lean operating level

Each plot depicts a superior system performance using the model-based control design. The model-based controller does not overshoot the desired point as significantly as the PI controller, it exhibits better dynamic control of the system, and it possesses an improved response time. However, because it was necessary to decrease the gains of the deadbeat controllers to avoid damaging the AD/DA converter and to limit the control effort to a .5 volt minimum to keep the burner lit, the error signal does not approach zero as quickly as the theoretical model and the response time is slightly longer than an ideal system.

In addition, the model-based controller exhibits a tendency to switch rapidly between control designs. This in turn causes a rapid variation in the carbon dioxide output in the exhuast, and is apperient during the first five seconds of the system's operation within each of the plots depicted above. Although this ringing will adversely affect the system's control components, it does not represent a significant problem. Filtering the control signal or employing " fuzzy" style logic within the control program should correct this behavior.

# VI SUMMARY

The successful formulation of the model-based controller represents a substantial step in the design of combustion control devices. Model-based approaches are inherently more accurate than the model-free approaches commonly used in most combustion applications, and provide significant benefits in the optimization of fuel economy or power. The system may be easily modified to sense carbon dioxide or oxygen in the exhaust gases, and control the system's air flow rather than its fuel feed. The design procedure developed within the project should be valid for the formulation of controllers capable of functioning from any system parameter.

The strength of the design undertaken within the project is its potential versatility. The controller is applicable to a limitless variety of problems, and any model-based design scheme may be used with the models derived. The deadbeat control designs presented provide a system capable, despite fluctuations in load and varying environmental conditions, of limiting the fuel feed into the combustion chamber to the narrow tolerances required for the desired mode of the reaction. The procedure for designing the controller should be compatible with any combustion system and sensor array. Either fuel-feed or airflow is capable of being regulated. The designer may select the parameters of the system's control, and the system models and controllers will be based upon these selections.

# REFERENCES


[1] K. J. Astrom and B. Wittenmark, *Computer Controlled Systems*. Englewood Cliffs: Prentice-Hall, 1984.

[2] K. J. Astrom and B. Wittenmark, *Adaptive Control*. New York: Addison Wesley, 1989.

[3] G. E. Box and G. M. Jenkins, *Time series analysis, forecasting and control*. San Francisco, CA: Holden-Day, 1970.

[4] S. Boyd, L. El Ghaoui, E Feron, and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory*. Philadelphia: SIAM, 1994.

[5] J. Chomiak, *Combustion; A Study In Theory, Fact, and Application*. New York: Gordon and Beach Science Publishers, 1990.

[6] C. Hang, K. Astrom, and W. Ho, "Refinements of the Ziegler-Nichols tuning formula," *IEEE Proceedings-D*, vol. 138, pp. 111-118, 1991.

[7] John D. Azzo, and Constantine Hoopis, *Linear Control System Analysis and Design*, New York: McGraw-Hill, 1988

[8] T. Johansen and B. A. Foss, "Constructing NARMAX models using ARMAX models," *International Journal of Control*, vol. 58, no. 5, pp. 1125-1153, 1993.

[9] J. Kocijan, "An approach to multivariable combustion control design," *Journal of Process Control*, vol. 7, no. 4, pp. 291-301, 1997.

[10] David Lindsley, *Boiler Control Systems*. New York: McGraw-Hill Book Company, 1991.

[11] Katsuhiko Ogata, *Modern Control Engineering*. Upper SaddleRiver, NJ: Prentice-Hall, 1997.

[12] Charles L. Phillips and H. Troy Nagle, *Digital Control System Analysis and Design*. Englewood Cliffs: Prentice-Hall, 1995.

[13] Giorgio Rizzoni, *Principles and Applications of Electrical Engineering*. Boston: Irwin, 1996.

[14] Richard Stone, *Introduction to Internal Combustion Engines*. Warrendale: Society of Automotive Engineers, Inc., 1992.

# APPENDIX A MODEL DESIGN PROGRAM

```
#include<stdio.h>
#include<dos.h>
#include<conio.h>
#include<math.h>
#include<float.h>
#include<time.h>
#include<fstream.h>
#include<iostream.h>
#include<iomanip.h>
#include <stdlib.h>
#include <time.h>

#define BASE 0x2E0                          // board base address

float ADCON_S ( char );                     // a/d conversion Sensor
void DACON( char, float);                   // d/a conversion routine
void wait(void);                            // wait routine

int main()
{
        float SenNumb= 2;                   //Sensor Voltage
        float y;

        float DANumb= 0;                    //Valve Control Voltage
        float u = 3;                        //Control Valve Starting Position

 int i;
 int j;

float uu = 0;
float uuu = 0;
float yy = 0;
float yyy = 0;
float yyyy = 0;
float yyyyy = 0;
float yyyyyy = 0;

 int h;
 int v;

 float g;                                   //increment tools
 float r;
 float c;
 float l;
 float li;
 float f;
 float n;
 float m;
```

```
int Time = 20; //sampling time

DACON(DANumb, u);

y = ADCON_S(SenNumb);

float phi[] = {yy, yyy, yyyy, yyyyy, yyyyyy, uu, uuu};        //phi vector defined

const int size = sizeof phi / sizeof phi[0];                  //Size of phi vector

float th[size][1];                                            //Initial theta matrix

float Kt[size][1];                                            //Kt matrix defined

float Zt[size][1];

float qT;

float Qt;

float A[1][size];

float S[1][size];

float R[size][size];

float b[size][1];

float Pt[size][size];

for(h = 0 ; h < size ; h++)
{
        Pt[h][h] = 10000;
}

for(h = 0 ; h < size ; h++)
{
        Kt[h][0] = 0;

        for(v = 0 ; v < size ; v++)
        {
                g = Pt[h][v]*phi[v];

                Kt[h][0] = Kt[h][0] + g;
        }
       }

for(j = 0 ; j < size ; j++)        //Calculation of th(to)
{
        th[j][0]=Kt[j][0] * y;
}
```

```
uuu = uu;
uu = u;
yyyyyy = yyyyy;
yyyyy = yyyy;
yyyy = yyy;
yyy = yy;
yy = y;

ofstream outfile1("C:\\Document\\outu.txt");
outfile1 << setw(10) << u;
outfile1<<endl;

ofstream outfile3("C:\\Document\\outy.txt");
outfile3 << setw(10) << y;
outfile3<<endl;

ofstream outfile2("C:\\Document\\outth.txt");

//End Initial Run

int t;                  //Sampling Time Delay

        for (t=0; t<Time; t++)
        {
        wait();
        }

 randomize();

//Calculation loop_____________________________________________
        while (kbhit() == 0)
        {
        float phi[] = {yy, yyy, yyyy, yyyyy, yyyyyy, uu, uuu};

        //Random Number Generator
                m = random(500);
                if (m < 500)
                        {
                                while (m < 500)
                                        {
                                                m = random(500)*random(500);
                                        }
                        }
                if (m > 500)
                        {
                                while (m > 500)
                                        {
                                                m = random(500) - random(100);
                                                if (m < 50)
                                                        {
                                                                while (m < 50)
                                                                        {
```

```
                m = random(500)*random(500);
                                                                }
                                                        }
                                        }
                        }

                n = 100;
                u = m/n;
                if (u < .5)
                        {
                                while (u < .5)
                                {
                                        u=random(5);
                                }
                        }

//End Number Generator

 DACON(DANumb, u);

 outfile1 << setw(10) << u;
 outfile1<<endl;

 outfile3 << setw(10) << y;
 outfile3<<endl;

        for(i = 0 ; i < 5 ; i++)
         {
                outfile2 << setw(15) << th[i][0];
         }
         outfile2 << endl;
        for(i = 5 ; i < size ; i++)
         {
                outfile2 << setw(15) << th[i][0];
         }
         outfile2 << endl;

y = ADCON_S(SenNumb);

for(h = 0 ; h < size ; h++)
        {
                Zt[h][0] = 0;

        for(int v = 0 ; v < size ; v++)
         {
                c = Pt[h][v]*phi[v];

                Zt[h][0] = Zt[h][0] + c;
         }
        }
```

```
float Wt = 0;

for( v = 0 ; v < size ; v++)            //Step 2
        {
                g = phi[v] * Zt[v][0];

                Wt = Wt + g;
        }

qT = Wt + 1;

Qt = 1/qT;

for(i = 0 ; i < size ; i++)
        {
                A[0][i] = 0;

                for(j = 0 ; j < size ; j++)
                 {
                        r = phi[j] * Pt[j][i];

                        A[0][i] = r + A[0][i];
                 }
        }

for(j = 0 ; j < size ; j++)
        {
                S[0][j] = Qt * A[0][j];
        }

for(v = 0 ; v < size ; v++)
        {
                for(j = 0 ; j < size ; j++)
                 {
                        R[v][j] = Zt[v][0]*S[0][j];
                 }
        }

for(i = 0 ; i < size ; i++)
        {
                for(v = 0 ; v < size ; v++)
                 {
                        Pt[i][v] = Pt[i][v] - R[i][v];
                 }
        }

for(h = 0 ; h < size ; h++)
        {
                Kt[h][0] = 0;

                for(int v = 0 ; v < size ; v++)
                 {
```

```
                    g = Pt[h][v]*phi[v];

                    Kt[h][0] = Kt[h][0] + g;
                }
        }

li=0;

for(h = 0 ; h < size ; h++)
        {
                l = phi[h] * th[h][0];

                li = li + l;
        }

f = y - li;

for(int j = 0 ; j < size ; j++)
        {
                b[j][0]=Kt[j][0] * f;
        }

for(v = 0 ; v < size ; v++)
        {
                th[v][0] = th[v][0] + b[v][0];
        }

uuu = uu;
uu = u;
yyyyyy = yyyyy;
yyyyy = yyyy;
yyyy = yyy;
yyy = yy;
yy = y;


int t;                                          //Sampling Time Delay

        for (t=0; t<Time; t++)
        {
        wait();
        }
        cout << setw(10) << u;
        cout << endl;
}
//LOOP End_____________________________________


outfile1 << setw(10) << u;
outfile1 << endl;
```

```
 outfile3 << setw(10) << y;
 outfile3<<endl;

 for(i = 0 ; i < 5 ; i++)
        {
                outfile2 << setw(15) << th[i][0];
        }
        outfile2 << endl;
 for(i = 5 ; i < size ; i++)
        {
                outfile2 << setw(15) << th[i][0];
        }
        outfile2 << endl;

for(i = 0 ; i < size ; i++)
{
cout <<  " + " << th[i][0];
}
return 0;
}

float ADCON_S( char SenNumb)                    // A/D Sensor conversion
        {
        int adin;high,low;
        int ReadBit=0x80;

        outportb(BASE+2, SenNumb);                      // Select a/d channel number
        outportb(BASE+1, 0);                            // Initiate a/d conversion
        while((inportb(BASE+2) & ReadBit) != 0 )        // wait until done
                {}
        high = inportb(BASE+1);                         // [b11,b10,...,b4]
        high = high << 4;                               // [b11,b10,...,b4,x,x,x,x]
        low = inportb(BASE);                            // [b3,b2,b1,b0,x,x,x,x]
        low = low >> 4;                                 // [x,x,x,x,b3,b2,b1,b0]
        low = low & 0x0f;                               // [0,0,0,0,b3,b2,b1,b0]
        adin = high | low;                              // [b11,b10,...,b1,b0]
        return( -5.0 + 0.002442*adin);          // convert to volts and return
        }                                               // 0.002442 = 10/4095

void DACON( char DANumb, float y) // D/A conversion
{
        int adout, high, low;
        int port;

        adout = (y + 5.0)*(409.5); // convert to 0<=adout<=4095
                                                        // adout = [b11,b10,...,b1,b0]
        port = BASE + 4 + 2*DANumb;                     // port address
        low = adout & 0x0ff;                            // [b7,b6,...,b1,b0]
        outportb(port,low);                             // write low byte
        high = adout >> 8;                              // [x,x,x,x,b11,b10,b9,b8]
        high = high & 0x0f;                             // [0,0,0,0,b11,b10,b9,b8]
        outportb(port+1,high);                          // write high byte
```

```
        inportb(BASE+3);                                    // initiate conversion
}
void wait(void)
{
        unsigned char bytein,previous;
        bytein = (inportb(BASE+3) & 0x01);                  // read io port bit 0, pin 24
        do      {                                           // digital ground pin 36
                previous = bytein;                          // current becomes previous
                bytein = (inportb(BASE+3) & 0x01);          // read io port bit 0
                }
        while(((bytein == 1)&&(previous == 0)) == 0);       // while no 0 to 1 transition
                {}                                          // wait here
}
```

# APPENDIX B PI CONTROL SCHEME

```
#include<stdio.h>
#include<dos.h>
#include<conio.h>
#include<math.h>
#include<float.h>
#include<time.h>
#include<fstream.h>
#include<iostream.h>
#include<iomanip.h>
#include <stdlib.h>
#include <time.h>
#define BASE 0x2E0                              // board base address

float ADCON_S ( char );                         // a/d conversion Sensor
void DACON( char, float);                       // d/a conversion routine
void wait(void);                                // wait routine
int main()
 {
 float SenNumb = 2;
 float DANumb = 0;
 float u_input;
 float y_error;
 float y_out;
 float y_outi = 0;
 float u_c;
 float u_ci = 0;
 int t;
 int Time = 20;

        printf("Enter Voltage Set Point \n");
        scanf("%f", u_input);

         while (kbhit() == 0)
                {
                y_error = ADCON_S(SenNumb);

                u_c = u_input - y_error;

                y_out = y_outi + u_c*9.59616 - u_ci*9.037332;

                if(y_out < .5)
                                {
                                DACON(DANumb, .5);
                                }
                        else
                                {
                                DACON(DANumb, y_out);
                                }
```

```
                y_outi = y_out;
                u_ci = u_c;
                for (t=0; t<Time; t++)
                        {
                        wait();
                        }
                }

        return 0;
        }
float ADCON_S( char SenNumb)                    // A/D Sensor conversion
{
        int adin,high,low;
        int ReadBit=0x80;
        outportb(BASE+2, SenNumb);              // Select a/d channel number
        outportb(BASE+1, 0);                    // Initiate a/d conversion
        while((inportb(BASE+2) & ReadBit) != 0 ) // wait until done
                {}
        high = inportb(BASE+1);                 // [b11,b10,...,b4]
        high = high << 4;                       // [b11,b10,...,b4,x,x,x,x]
        low = inportb(BASE);                    // [b3,b2,b1,b0,x,x,x,x]
        low = low >> 4;                         // [x,x,x,x,b3,b2,b1,b0]
        low = low & 0x0f;                       // [0,0,0,0,b3,b2,b1,b0]
        adin = high | low;                      // [b11,b10,...,b1,b0]
        return( -5.0 + 0.002442*adin);          // convert to volts and return
}                                               // 0.002442 = 10/4095
void DACON( char DANumb, float y) // D/A conversion
{
        int adout, high, low;
        int port;
        adout = (y + 5.0)*(409.5);                              // convert to 0<=adout<=4095
                                                                // adout = [b11,b10,...,b1,b0]
        port = BASE + 4 + 2*DANumb;                             // port address
        low = adout & 0x0ff;                                    // [b7,b6,...,b1,b0]
        outportb(port,low);                                     // write low byte
        high = adout >> 8;                                      // [x,x,x,x,b11,b10,b9,b8]
        high = high & 0x0f;                                     // [0,0,0,0,b11,b10,b9,b8]
        outportb(port+1,high);                                  // write high byte
        inportb(BASE+3);                                        // initiate conversion
}
void wait(void)
{
        unsigned char bytein,previous;
        bytein = (inportb(BASE+3) & 0x01);              // read io port bit 0, pin 24
        do      {                               // digital ground pin 36
                previous = bytein;              // current becomes previous
                bytein = (inportb(BASE+3) & 0x01);      // read io port bit 0
                }
        while(((bytein == 1)&&(previous == 0)) == 0);  // while no 0 to 1 transition
                {}                              // wait here
}
```

# APPENDIX C NON-LINEAR CONTROL SCHEME

```
#include<stdio.h>
#include<dos.h>
#include<conio.h>
#include<math.h>
#include<float.h>
#include<time.h>
#include<fstream.h>
#include<iostream.h>
#include<iomanip.h>
#include <stdlib.h>
#include <time.h>

#define BASE 0x2E0                          // board base address

float ADCON_S ( char );                     // a/d conversion Sensor
void DACON( char, float);                   // d/a conversion routine
void wait(void);                            // wait routine

int main()
 {
 float SenNumb = 2;
 float DANumb = 0;

 float u_input;
 float y_error;
 float y_out;
 float y_outi = 0;
 float y_outii = 0;
 float y_outiii = 0;
 float y_outiiii = 0;
 float y_outiiiii = 0;
 float u_c;
 float u_ci = 0;

 int t;
 int Time = 20;

        printf("Enter Voltage Set Point \n");
        scanf("%f", u_input);

         while (kbhit() == 0)
                {
                y_error = ADCON_S(SenNumb);

                u_c = u_input - y_error;

                if(y_error > .6)
                        {
y_out = .993659*y_outi-.0105317*y_outii-.00546735*y_outiii+.0183735*y_outiiii -
```

```
        +.00321175*y_outiiiii+.000969177*u_c-.000707305*u_ci;
                        }
                else if(y_error < .4)
                        {
y_out = .955412*y_outi+.0246399*y_outii+.00105189*y_outiii+.00338509*y_outiiii
        +.0138404*y_outiiiii+.00047797*u_c-.0000092*u_ci;
                        }
                else
                        {
y_out = .961226*y_outi+.0209421*y_outii+.00112333*y_outiii+.0174243*y_outiiii
        -.0146176*y_outiiiii+.00172242*u_c+.00146233*u_ci;
                        }

                        if(y_out < .5)
                                {
                                DACON(DANumb, .5);
                                }
                        else
                                {
                                DACON(DANumb, y_out);
                                }

                y_outi = y_out;
                y_outii = y_outi;
                y_outiii = y_outii;
                y_outiiii = y_outiii;
                y_outiiiii = y_outiiii;

                u_ci = u_c;

                for (t=0; t<Time; t++)
                        {
                        wait();
                        }
                }

         return 0;
        }

float ADCON_S( char SenNumb)                    // A/D Sensor conversion
{
        int adin,high,low;
        int ReadBit=0x80;

        outportb(BASE+2, SenNumb);              // Select a/d channel number
        outportb(BASE+1, 0);                    // Initiate a/d conversion
        while((inportb(BASE+2) & ReadBit) != 0 )  // wait until done
                {}
        high = inportb(BASE+1);                 // [b11,b10,...,b4]
        high = high << 4;                       // [b11,b10,...,b4,x,x,x,x]
        low = inportb(BASE);                    // [b3,b2,b1,b0,x,x,x,x]
        low = low >> 4;                         // [x,x,x,x,b3,b2,b1,b0]
```

```
        low = low & 0x0f;                       // [0,0,0,0,b3,b2,b1,b0]
        adin = high | low;                      // [b11,b10,...,b1,b0]
        return( -5.0 + 0.002442*adin);          // convert to volts and return
}                                               // 0.002442 = 10/4095

void DACON( char DANumb, float y) // D/A conversion
{
        int adout, high, low;
        int port;

        adout = (y + 5.0)*(409.5);      // convert to 0<=adout<=4095
                                        // adout = [b11,b10,...,b1,b0]
        port = BASE + 4 + 2*DANumb;     // port address
        low = adout & 0x0ff;            // [b7,b6,...,b1,b0]
        outportb(port,low);             // write low byte
        high = adout >> 8;              // [x,x,x,x,b11,b10,b9,b8]
        high = high & 0x0f;             // [0,0,0,0,b11,b10,b9,b8]
        outportb(port+1,high);          // write high byte
        inportb(BASE+3);                // initiate conversion
}

void wait(void)
{
        unsigned char bytein,previous;
        bytein = (inportb(BASE+3) & 0x01);      // read io port bit 0, pin 24
        do      {                               // digital ground pin 36
                previous = bytein;              // current becomes previous
                bytein = (inportb(BASE+3) & 0x01);      // read io port bit 0
                }
        while(((bytein == 1)&&(previous == 0)) == 0);   // while no 0 to 1 transition
                {} }                            // wait here
```

# APPENDIX D

## PHYSICAL DESIGN

### Exhaust Flue

The exhaust flue and first cooling tank are constructed of 1/8 inch stainless steel. The second cooling tank is plastic, and rests upon a 10 ¾-inch high wooden platform. The legs of the flue assembly stand 9 ¼ inches high, and allow the burner and cuff assembly to be bolted within the flue. This measure ensures that the burner is isolated from ambient oxygen. The flue assembly is 5 inch square and is curved in two 180 degree elbows. The water filled cooling tanks are situated at each elbow to decrease the temperature of the exhaust. These tanks were used to vary the load characteristics of the system during its operation. The total height of the experimental apparatus is 38 inches and the 'S' curves generate an overall width of 18 ¼ inches. Figures 1a and b depict various views of the exhaust flue and cooling tanks.

### Burner Assembly

As shown in figures 1a and b, the burner assembly is composed of a Bunsen Burner that has been modified to prohibit ambient air from being introduced into the system. The cuff of the burner assembly is manufactured from copper tubing, and is 1 ½ inches wide. The cuff is silver soldered to a 1/8 inch thick steel plate that is 5 inch square, and is used to secure the burner assembly within the flue. A ½ inch copper tubing 'S' curve connects the air supply to the assembly. Silicone caulking is used to seal all the burner assembly joints that are not welded or soldered.

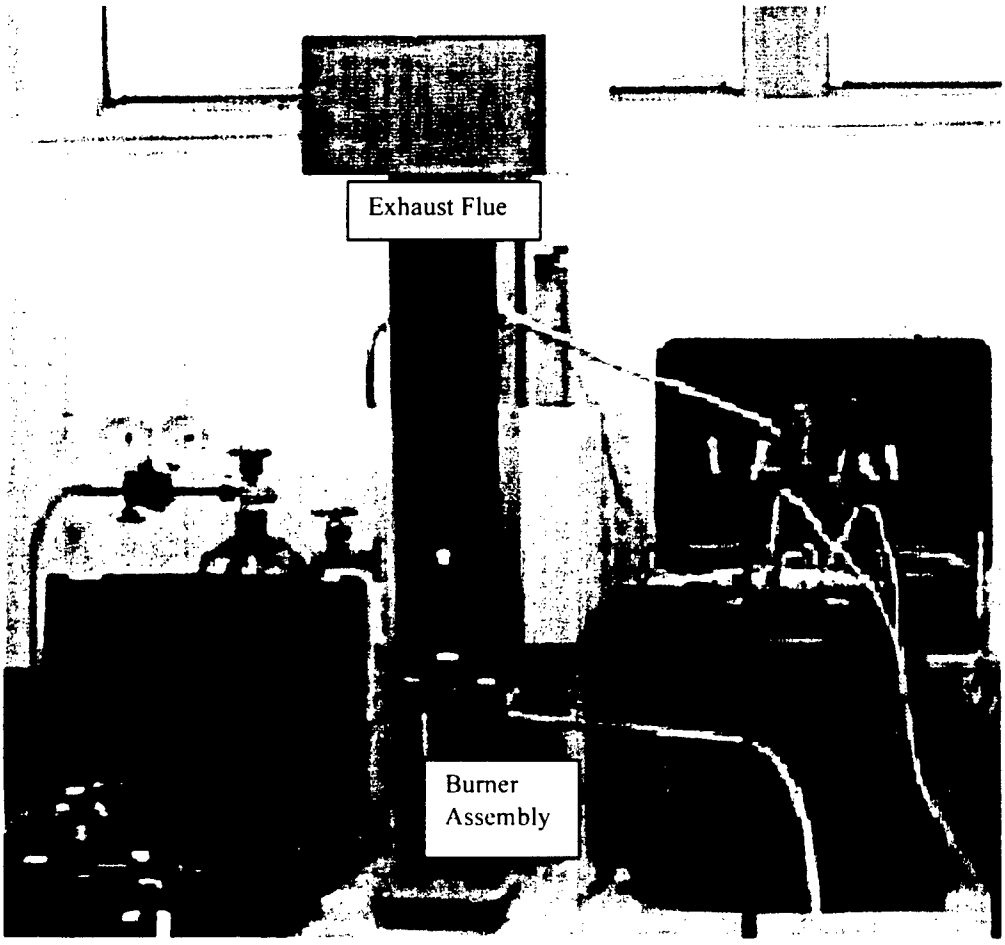



Figure 1a Photograph of Exhaust Flue and Burner Assembly

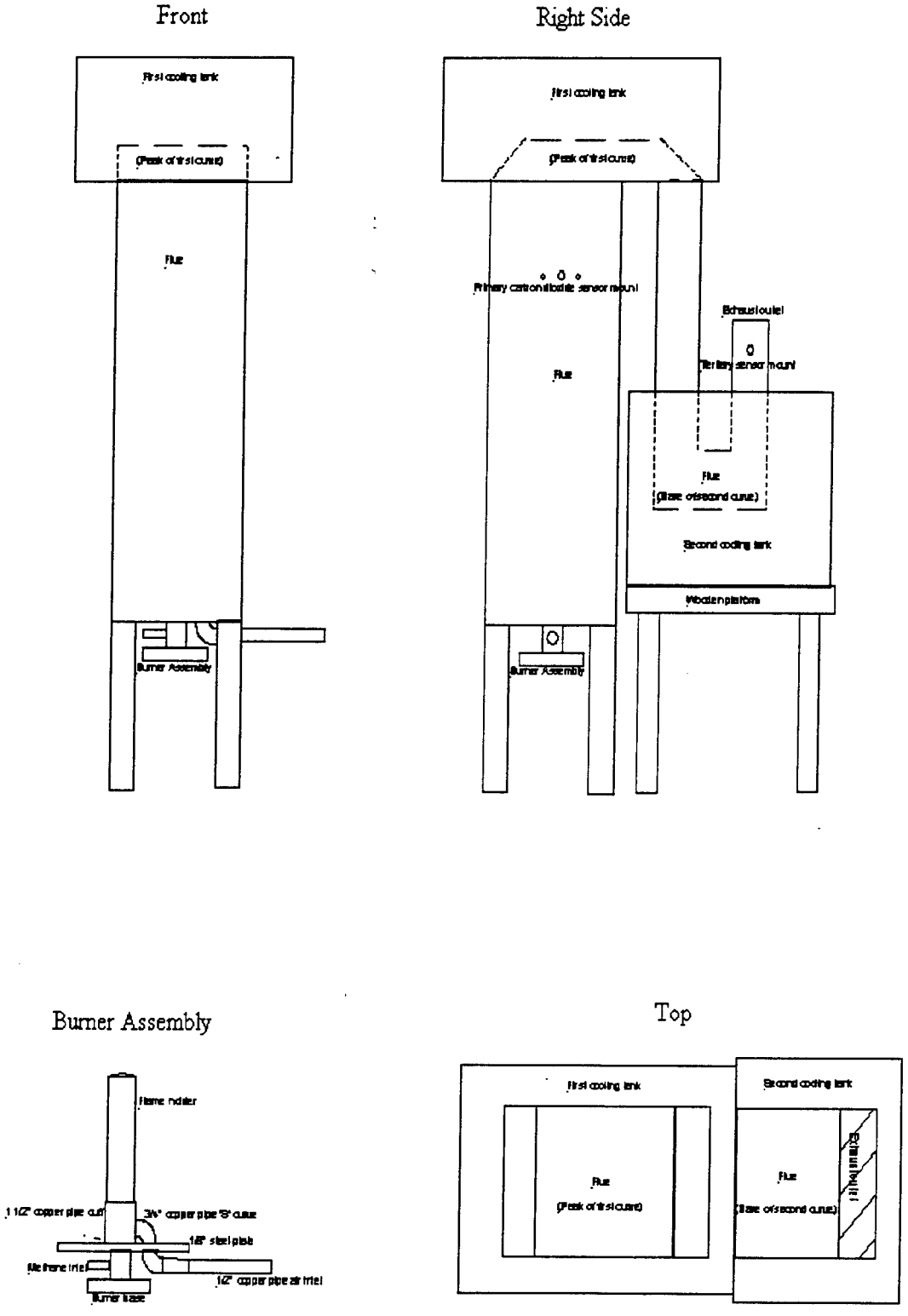



Figure 1b Construction Diagrams of Exhaust Flue and Burner

## Fuel and Air Sources

The experimental apparatus is fueled by compressed methane. Gas is supplied from the cylinder to the burner through ¼'' surgical tubing. A solenoid valve controlled by a personal computer is used to regulate the methane flow to the burner, and a gas flow meter in series with the valve registers the volumetric gas flow. Figure 2 depicts the fuel supply.

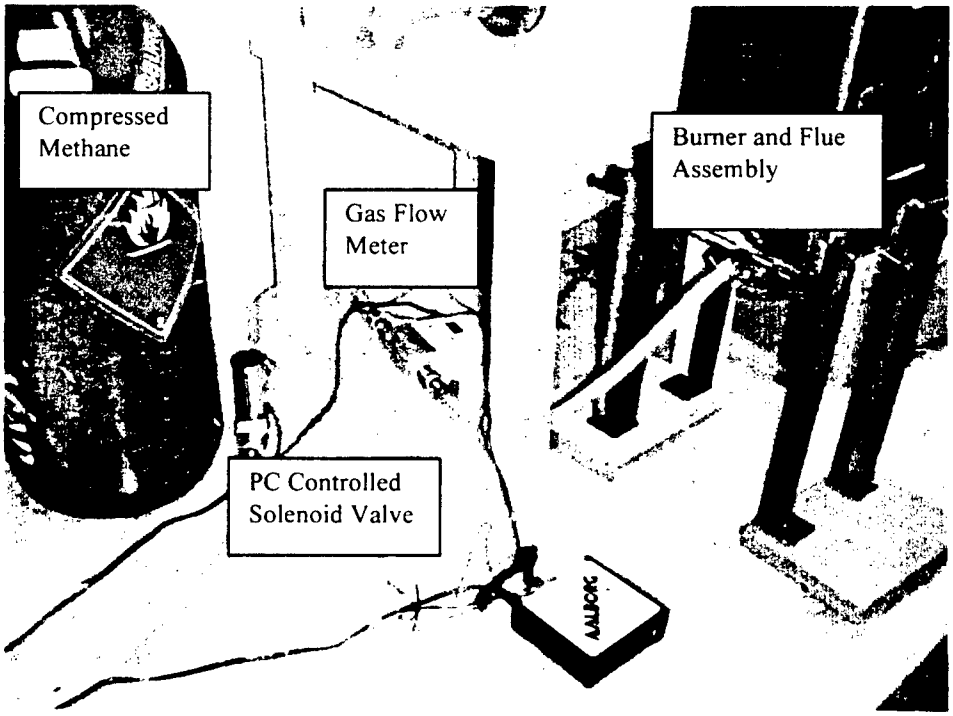



Figure 2 Fuel Source

Air is supplied by the compressor pictured in Figure 3. A solenoid valve, in series with a gas flow meter, is used to control manually the amount of air reaching the burner assembly. Through the solenoid valve and gas flow meter, a 1 inch plastic hose connects the compressor to the burner. This source of air is an improvement over the fan that was used during the early stages of the experiment. Since the solenoid valve that now regulates the flow from the compressor was not initially available, a fan was used to provide the necessary air. An aluminum reducer was manufactured to duct the air supplied by the fan into the plastic tube connected to the burner. The air introduced into the system through this earlier method could not be

regulated. The fan assembly was only capable of providing a fixed flow, and this flow could not be measured because the pressure generated by the fan was insufficient to allow the gas flow meter to be used. Measurements of the air flow were necessary during the final stages of the experiment in order to codify the operating points of the burner.

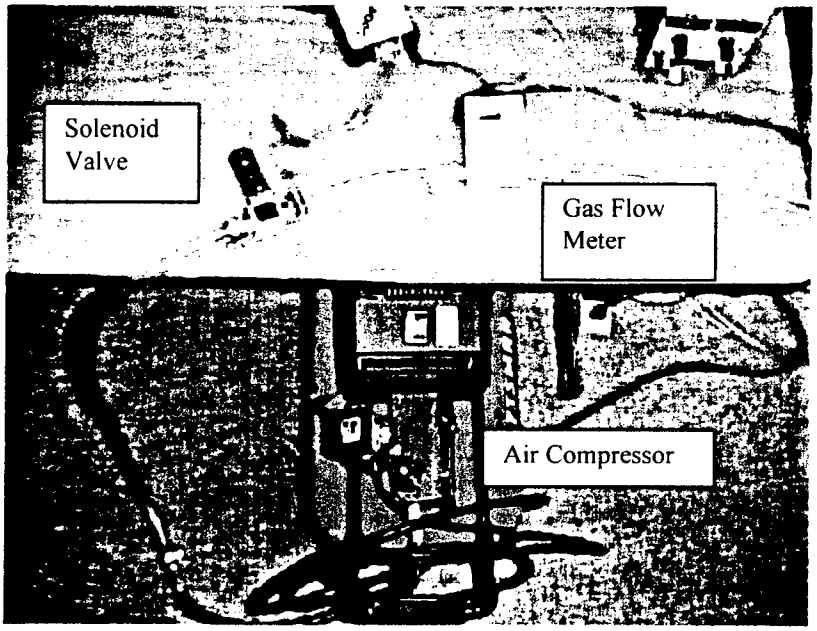



Figure 3 Air compressor, solenoid valve, and gas flow meter in series

# APPENDIX E

Taking the Laplace Transform followed by the Z-Transform of equation 3.1, the following equation may be derived,

$$D(z) = K_c + \frac{K_c}{T_i}\frac{T}{2}\left[\frac{z+1}{z-1}\right] \tag{1}$$

where T is the sampling period. This is a digital PI filter that utilizes trapezoidal integration. Further simplifying the equation yields the digital transfer function:

$$D(z) = \frac{\frac{2K_cT_i + K_cT}{2T_i} + \frac{K_cT - 2K_cT_i}{2T_i}z^{-1}}{1 - z^{-1}} \tag{2}$$

Before the equation may be implemented into the control program, the digital transfer function must be converted into a difference equation. Equation 2 yields:

$$E(z) - z^{-1}E(z) = \left(\frac{2K_cT_i + K_cT}{2T_i}\right)U(z) + \left(\frac{K_cT - 2K_cT_i}{2T_i}\right)z^{-1}U(z)$$

$$D(z) = \frac{E(z)}{U(z)} \tag{3}$$

Equation 3 represents the Z-transform of the difference equation. E(z) is the control effort output, and U(z) is the error signal input. The final form of the equation, which may be used in the digital control program is:

$$e(k) = e(k-1) + \left(\frac{2K_cT_i + K_cT}{2T_i}\right)u(k) + \left(\frac{K_cT - 2K_cT_i}{2T_i}\right)u(k-1) \tag{4}$$

The terms containing a negative exponent, and those including the statement (k-1) represent delays. They designate data that was gathered during previous sampling periods.